

**Study Book For Computer
Science, According To
National Curriculum 2009**

**Fundamentals of
Operating System,
SDLC & C++**

F.M SERIES

**GRADE
12**

**SHIEKH
FAISAL
MANZOOR**

All Right Reserves to the Author Owner

F.M SERIES.

FUNDAMENTALS OF OPERATING SYSTEM, SDLC & C++

For GRADE 12

Written By

Sheikh Faisal Manzoor

Head of Computer Science Dept:

FG Degree College Quetta Cantt

Acknowledgements:

1. Mr. Muhammad Khalid, Computer Programmer, Balochistan Board of Intermediate and Secondary Education Quetta.
2. Syed Hassan Faraz, Assist. Prof of Computer Science General Musa Govt Degree College Quetta.
3. Mr. Tulja Ram, Assist. Prof of Computer Science Girls FGDC Quetta.
4. Mr. Shahid Barket, Lecturer in Computer Science Islamia Boys College Quetta.
5. Mr. Baqir Raza, Lecturer in Computer Science Gen Musa Govt Degree College Quetta.
6. Muhammad Waqas, Lecturer in Computer Science Govt Boys Degree College Zhob.
7. Mr. Nawab Din, Lecturer in Computer Science Govt Boys Degree College Turbat.
8. Mr. Abdullah Sumro, Lecturer in Computer Science Girls FGDC Quetta.
9. Mr. Rashid Ahmed, Lecturer in Computer Science Govt Girls Degree College Quwary road Quetta.

Published by: Sheikh Faisal Manzoor

Edition: 1st Edition 2020

All Rights Reserved with the publisher. This book, in whole or in part, may not be reproduced, stored in a retrieval system, or transmitted, in any form or by any means in any language or otherwise, without prior written permission of the publisher.

PREFACE

Computer knowledge like all scientific knowledge has maintained its importance in this modern age because of its utility in every day dealings. This branch of knowledge has shown its impact on almost all modern fields of research.

I thank Almighty God, who has given me ability to frame this study book to help the student of XII year studying Computer Science. It is in accordance with the latest National Curriculum for Computer Science (2009).

I have framed this book with sheer sincerity and utmost effort to help the students in achieving their objectives. I have tried to put the matter in an easy language. An important feature of this Study Guide is that it covers every topic including Operating System, System Development Life Cycle, Object Oriented Programming in C++, Control Structures, Arrays and Strings, Functions, Pointers, Object and Classes and File Handling. Solved exercise at the end of every chapter and important abbreviations at the end of this book have been given for further guidance. Following sources and books are consulted for the preparation and compilation of this guide.

- Turbo C++ by example
- Internet

I do hope this study guide will equip the students with a wide range of computer knowledge and win favour and admiration from the students. Any suggestion for its further improvement is however invited.

FAISAL MANZOORTM

Head of Computer Science Dept: F.G Degree College Quetta Cantt

Ex-LAN Administrator SBK Women's University Quetta.

Test Administration Staff in National Testing Service of Pakistan.

Exhibitor in ITCN Asia 2004

E-mail: misterfaisal.fg@gmail.com, misterfaisal@hotmail.com

Face Book ID: Sheikh Faisal Manzoor Cell: 03337926605

Youtube channel link: <https://www.youtube.com/channel/UCAuBv05i0IGaHNFwT9s1z8A>

INDEX

Unit 1: OPERATING SYSTEM 6

- 1.1 Introduction to Operating System 6
- i) Define an operating system 6
- ii) Describe commonly-used operating systems (DOS, Windows, Unix, Macintosh) 7,8
- Comparison between DOS and Windows 8
- iii) Explain the following types of operating system:
- ☒ Batch processing Operating System 9
 - ☒ Multi-programming Operating System 9
 - ☒ Multi-tasking Operating System 9
 - ☒ Time-Sharing Operating System 9
 - ☒ Real-Time Operating System
 - ☒ Multi-processor Operating System 10
 - ☒ Parallel Processing Operating Systems 10
 - ☒ Distributed Operating Systems 10
 - ☒ Embedded Operating System 10
- iv) Define the following features/characteristics of operating system:
- ☒ Single-user Operating Systems 10
 - ☒ Multi-user Operating System 11
- 1.2 Operating System Functions
- Describe the following main functions of operating system:
- ☒ Process Management 11
 - ☒ Memory Management 11
 - ☒ File Management 12
 - ☒ I/O System Management 12
 - ☒ Secondary Storage Management 12
 - ☒ Network Management 13
 - ☒ Protection System 13
 - ☒ Command-Interpreter 13
- 1.3 Process Management 13
- i) Define a process 13
- ii) Describe the new, running, waiting/blocked, ready and terminated states of a process 13
- iii) Differentiate between:
- ☒ Thread and process 14
 - ☒ Multi-threading & multi-tasking 14
 - ☒ Multi-tasking and multi-programming 15
- EXERCISE 15**

Unit 2: SYSTEM DEVELOPMENT LIFE CYCLE 19

- 2.1 System Development Life Cycle
- i) Define a System 19
- ii) Explain System Development Life Cycle (SDLC) and its importance 19
- iii) Describe objectives of SDLC 19
- iv) Describe stakeholders 20
- v) Explain the following:

- ☐ Planning 20
 - ☐ Feasibility 21
 - ☐ Analysis 21
 - ☐ Requirement Engineering 21
 - Requirement Gathering 21
 - Functional Requirements 22
 - Non Functional Requirements 22
 - Requirements Validation 22
 - Requirements Management 22
 - Design (Algorithm, Flow Chart) 22
 - Coding 24
 - Testing/verification 24
 - Deployment/Implementation 25
 - Maintenance/Support 25
 - vi) Explain the role of following stakeholders / personals in the system development life Cycle:
 - Management 26
 - Project Manager 26
 - System Analyst 26
 - Programmer 27
 - Software Tester 27
 - Customer 27
- EXERCISE 27**

Unit 3: OBJECT ORIENTED PROGRAMMING IN C++ 30

- 3.1 Introduction
- i) Define Program, source code, object code 30
- C++, IDE 30
- Dev C++ IDE, Steps of creating program in dev C++ 31
- ii) Define header files and reserved words
- iii) Describe the structure of a C++ program 33
- ☒ Pre-processor Directives 33
 - include 33
 - ☒ Main function 33
 - ☒ Body 33
- iv) Know the use of a statement terminator (;)
- 3.2 C++ Constants and Variables
- i) Constant 34
- ii) Variable and the rules for specifying variable names, declaring and initializing variable 35
- iii) Know the following data types offered by C++ and the number of bytes taken by each data type 36
- ☒ Integer - int (unsigned, short, long)
 - ☒ Floating point - float
 - ☒ Double precision - double
 - ☒ Character - char
- iv) Define constant qualifier - const 36
- 3.3 Input/ Output Handling 37
- i) Explain the use of cout statement for displaying output on the screen 37

ii) Explain the use of `cin` statement to get input from the keyboard during execution of the program 40

iii) Define `getch()`, `gets()` and `puts()` functions 38

iv) Define escape sequence 38

v) Explain use of the following escape sequences using programming examples 38

☐ Alert - \a

☐ Backspace - \b

☐ Newline - \n

☐ Carriage Return - \r

☐ Tab - \t

☐ Display backslash - \\

☐ Display single quotation marks - \'

☐ Display double quotation mark - \"

vi) Use manipulators `endl` and `setw` 38

3.4 Operators in C++ i) Define the following operators and show their use with examples: 41

☐ Assignment operator (=) 41

☐ Arithmetic operators (+, -, *, /, %) 41

☐ Arithmetic assignment operators(+ =, -=, *=, /=, %=) 42

☐ Increment and decrement operators (++ , --) 44

- Prefix

- Postfix

☐ Relational operators (<, >, <=, >=, ==, !=) 43

☐ Logical operators (AND, ||, OR &&, NOT !)

☐ Ternary operator (? :) 45

ii) Identify unary, binary and ternary operators 47

Comments, single line, multi line 48

EXERCISE 48

Unit 4: CONTROL STRUCTURES 54

4.1 Control structures 54

i) Explain the use of the following decision statements:

☒ If 54

☒ If-else 55

☒ Else-if 57

☒ Switch-default 59

ii) Know the concept of nested if 56

Difference between else if and switch 61

4.2 Loops i) Explain the use of the following looping structures:

☒ For 61

☒ While 64

☒ Do-while 65

Difference between while and do while 68

ii) Break and continue statement, `exit()` function 67

iii) Know the concept of nested loop 68

EXERCISE 71

Unit 5: ARRAYS AND STRINGS 80

5.1 Introduction 80

i) Explain the concept of an array 80

ii) Know how array elements are arranged in memory 80

iii) Explain the following terms related to arrays

☐ Size of array 80

☐ Name of array 80

☐ Index 80

iv) Explain how to define and initialize an array of different sizes and data types 80

v) Explain how to access and write at an index in an array 81

vi) Use the `sizeof()` function to find the size of an array 84

5.2 Two dimensional Arrays 84

i) Explain the concept of a two dimensional array 84

ii) Explain how to define and initialize a two dimensional array of different sizes and data types

iii) Explain how to access and write at an index in a two dimensional array 85

5.3 Strings 87

i) Explain what are strings. 87

ii) Explain how to define a string 88

iii) Explain the techniques of initializing a string

iv) Explain the most commonly used string functions 88

EXERCISE 88

Unit 6: FUNCTIONS 100

6.1 Functions

i) Explain the concept and types of function 100

ii) Explain the advantages of using functions 100

iii) Explain the following terms related to functions

☐ Function prototype 101

☐ Function definition 101

☐ Function call 101

v) Explain the difference between local, global, and static variables 102, 103

vi) Explain the difference between formal and actual parameters 109

vii) Know the concept of local and global functions 106

viii) Use inline functions 107

6.2 Passing arguments and returning values

i) Pass the arguments:

☐ Constants 110

☐ By value 110

☐ By reference 111

ii) Use default argument 108

iii) Use `return` statement 106

6.3 Function overloading

i) Define function overloading	112
ii) Know advantages of function overloading	113
iii) Understand the use of function overloading with:	
□ Number of arguments	112
□ Data types of arguments	112
□ Return types	113
Difference between Local variable and global variable.	113
Difference between pass by value and pass by reference	114
EXERCISE	114

Unit 7: POINTERS 119**7.1 Pointers**

i) Define pointers	119
ii) Understand memory addresses	119
iii) Advantage of pointers	119
iv) Pointer declaration	119
v) Reference operator	120
v) Pointer initialization, void pointer	120
vi) void pointer, de-reference operator	120
EXERCISE	121

Unit 8: OBJECTS AND CLASSES 123**8.1 Classes**

i) Define class	123
ii) Know the member of a class:	
□ Data member	123
□ Member functions	123

iii) Understand and access specifier:	124
□ Private	124
□ Public	125
iv) Know the concept of data hiding	125
v) Define constructor and destructor	126
□ Default constructor/destructor	127
□ User defined constructor	127
□ Constructor overloading	127
vii) Understand the concept of following only with daily life examples:	
□ Inheritance	127
□ Polymorphism	129
EXERCISE	130

Unit 9: FILE HANDLING 135**9.1 File**

i) Know the binary and text file	135
ii) Open the file	
□ Modes of opening file	136
iii) Know the concept of	
□ BOF	139
□ EOF	139
iv) Define stream	139
v) Use the following streams	
□ Single character	140
□ String	142
EXERCISE	143

ANSWERS OF MCQs 145**IMPORTANT ABBREVIATIONS 146****NATIONAL CURRICULUM 2009 147**

UNIT 1

OPERATING SYSTEM

OPERATING SYSTEM

An Operating System is a type of system software that manages all the operations and computer's resources such as CPU, memory, disk drive etc. It also controls the execution of application programs and acts as an interface between the user of a computer and computer hardware. A computer cannot perform any task without an operating system. In every computer like desktop computer, tablet and smart-phone etc the operating system must be installed that provides basic functionality for the device. Some examples of operating system are as follows:

- Microsoft Windows
- Linux
- Unix
- Mac OS
- Android

TASKS PERFORMED BY OPERATING SYSTEM

Following are the important tasks performed by an operating system:

- 1- It loads programs into main memory and executes them.
- 2- It controls the operations of all the devices for example main memory and external storage etc.
- 3- It manages files and folders on storage device such as hard disk and USB flash drive etc.
- 4- It allows multitasking to execute multiple tasks at the same time such as running spreadsheet software and a word-processor simultaneously.
- 5- It performs networking operations that allow the user to communicate over the network and share the resources such as hard disk and printer etc.
- 6- It maintains the security of the computer through username and password.
- 7- It identifies the hardware problems and shows the messages to solve them.

COMMONLY USED OPERATING SYSTEMS

Following are the commonly used operating systems:

1- DOS

DOS stands for Disk Operating System. It was developed in 1970s when microcomputer was introduced. It was called Disk Operating System because the entire operating system could be stored on a single floppy disk. The user has to type the commands to interact with computer because it has command line interface (CLI). It was difficult for the beginners to learn, memorize and use these commands. Following are some of the DOS commands:

CD	It is used to change directory.
RENAME	It is used to change the name of the file.
DIR	It is used to display all files in a directory.
COPY	It is used to copy file from one drive/directory to another.

DEL

It is used to delete one or more files.

FORMAT

It is used to format the disk.

The interface of DOS is as follows:

```
C:\>dir

Volume in drive C is MS-DOS 5.0
Volume Serial Number is 446B-27B1
Directory of C:\

COMMAND  COM      47845 11-11-91  5:00a
          1 file(s)  47845 bytes
                   10280960 bytes free

C:\>ver

MS-DOS Version 5.00

C:\>
```

2- WINDOWS OPERATING SYSTEM

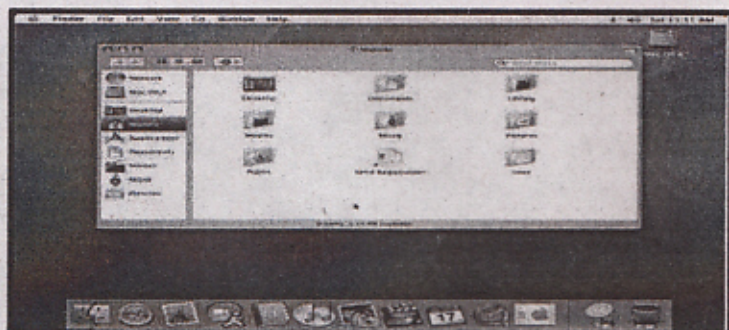
Windows is the most commonly used operating system. It was developed in mid 1980s by Microsoft Corporation. The user does not have to memorize the commands like DOS because it provides Graphical User Interface (GUI) to interact with computer. It allows user to give commands to computer through icons, menus, and buttons etc. Microsoft released many versions of Windows that includes Windows 95, Windows 98, Windows Millennium, Windows XP and Windows Vista etc. Windows 10 is the most recent operating system. The interface of Windows 10 is as follows:



3- MAC OS

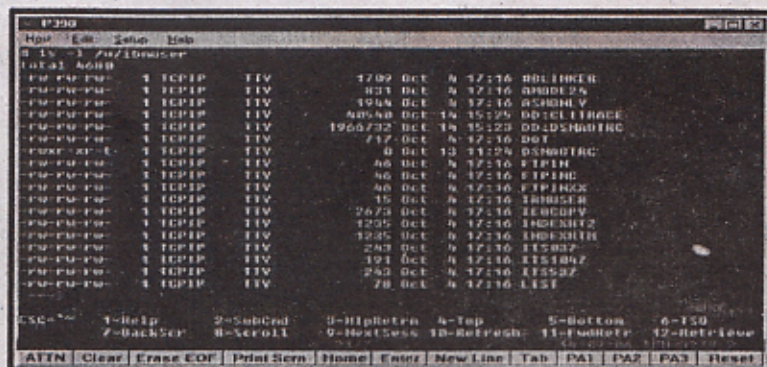
Mac OS is a series of operating systems developed by Apple Inc. Mostly it is installed on Apple computers like iMac, MacBook Pro, and MacBook Air. The latest version of Mac OS is known as OS X. It is a more secure operating system compared to Windows. Mac hardware and software work together with minimum problems. However, Mac OS is not widely used operating system as Windows. A large variety of application software is easily

available for Windows operating system whereas the OS X has very limited application software. Mac OS X interface is as follows:



4- UNIX

UNIX OS was developed in early 1970s at Bell Laboratories by Ken Thompson and Dennis Ritchie. It was developed in C-Language. It provides greater processing power and better security than windows OS. The computer running UNIX operating system usually do not get malware attach such as viruses. Many versions of UNIX are available for a wide range of computer systems from microcomputer to mainframe. It is less popular on microcomputer on which windows is pre-installed when they are sold. It is most commonly used in servers. The interface of UNIX OS is as follows:



COMPARISON BETWEEN DOS AND WINDOWS OPERATING SYSTEM

DOS	WINDOWS
1- It provides Command Line Interface.	1- It provides Graphical User Interface.
2- It is difficult to learn.	2- It is easy to learn.
3- It supports single tasking.	3- It supports multi-tasking
4- It is a single-user operating system.	4- It is a multi-user operating system.
5- It does not provide plug n play facility.	5- It provides plug n play facility.
6- It supports the use of only keyboard.	6- It supports the use of Keyboard & mouse.

TYPES OF OPERATING SYSTEM

Following are the types of operating systems that are commonly used on various computer systems

1- BATCH PROCESSING OPERATING SYSTEM

The batch processing operating system executes a set of same type of jobs automatically. A set of same types of jobs is known as a **batch**. All jobs in a batch are executed one after the other i.e when a job is completely finished then the next job is going to be started. The batch processing is an easy and efficient way of processing the same type of jobs. It is very useful for the tasks that require similar processing. For example the banks can use batch processing to print the bank statement of each account holder. The batch processing is normally performed using powerful mainframe computers.

2- MULTI-PROGRAMMING OPERATING SYSTEM

The multiprogramming operating system keeps multiple programs (jobs) in main memory at a time and executes them using a single CPU. It executes only one program (job) at a time while the other programs (jobs) are waiting in a queue. During execution, some jobs may need to wait for certain tasks (such as I/O operations). In batch processing system the processor would sit idle, however in multi-programming system, the CPU switches to second job and begins to execute it. In this way, multiprogramming OS uses CPU time and other computer resources to improve the performance.

3- MULTI-TASKING OPERATING SYSTEM

The multitasking operating system allows the user to perform multiple tasks at the same time on the computer system with single CPU. For example the user can work on word document while a webpage is being loaded in the browser. A single CPU actually cannot execute more than one task at a time but it rapidly switches between multiple programs and it looks that all the tasks are being executed at the same time.

4- TIME-SHARING OPERATING SYSTEM

The time-sharing OS are typically used in minicomputers and mainframe computers that supports large number of users. In time sharing operating system the CPU time is shared between multiple programs that are loaded in the main memory. It gives a very short period of CPU time to each program one by one. The time period is known as **time slice or time quantum**. The program is executed for the allocated time quantum and when the time quantum is over the control is transferred to the next program. The program waits for the next time quantum if it is not finished. All programs are executed in a cycle. The switching of CPU is so quick that each user thinks that computer is only performing his job.

The time-sharing OS are used in large organizations like airline, bank, university etc.

5- REAL-TIME OPERATING SYSTEM

A real time operating system executes a job within a specified time period. It completes the given task and responds very quickly. The real time operating systems are designed for very complex tasks. They are used in the fields where a quick response is very much important. Some real time applications are as follows:

- Airplane guidance system.
- Traffic control system

- Industrial system such as oil refinery.
- Monitoring the position of a rocket in the space.
- Airbag controls in cars.

6- MULTI-PROCESSING OPERATING SYSTEM

The multi-processing operating system controls the operations of two or more CPUs within a single computer system. All the CPUs of a computer share the same main memory and input output devices. Multiprocessing OS are used to process a large amount of data at very high speed. The computer needs complex architecture to support multiprocessing.

7- PARALLEL PROCESSING OPERATING SYSTEM

The parallel processing operating system supports the execution of a program on multiple processors at the same time. It divides big task into many smaller tasks that are executed on different processors simultaneously. It is suitable for the large and complex programs that require too much calculation at the same time. These operating systems are used in supercomputers that have thousands of processors.

8- DISTRIBUTED OPERATING SYSTEM

The distributed operating system manages the operations of distributed system. A distributed system is used to execute the program on different computers in a network. The program may run on one computer and access data from another computer in the network. The distributed operating system automatically distributes the tasks and balances the load among different computers to execute the programs efficiently.

9- EMBEDDED OPERATING SYSTEM

The embedded operating system is an operating system that is embedded (fixed) into the hardware. It controls the operations of devices such as microwave oven, TV, camera, washing machine, games etc. It runs automatically and performs the specific task when the device is turned on.

TYPES OF OPERATING SYSTEM BASED ON NUMBER OF USERS

The operating system can be divided into two types based on the number of users they can support. These are as follows:

1- SINGLE-USER OPERATING SYSTEM

The operating system that allows only one person to operate the computer at a time is known as single user operating system. DOS is an example of single user operating system. It is typically used on microcomputers.

2- MULTI-USER OPERATING SYSTEM

The operating system that allows many users on different computers to use the resources of single central computer (server) in a network is known as multi-user operating system. It manages the tasks of all the users efficiently. It requires powerful process, large memory and storage capacity. Some examples of multi-user operating system are UNIX, Linux and Windows server 200 onward and Max OS X.

FUNCTIONS PERFORMED BY OPERATING SYSTEM

Following are the functions performed by operating system

1- PROCESS MANAGEMENT

A program in execution is known as a process. A computer can execute many processes at a time. A process may need resources such as processor, memory and input/output. The operating system handles these tasks for the processes. It allocates the required resources to the process. It also protects the resources of one process from other processes. Process management actually describes the state and resource ownership of each process.

Example: Suppose three processes X, Y and Z are ready for execution as follows:

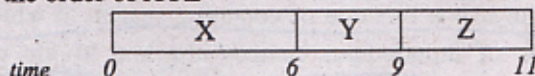
Process X Suppose it has the CPU cycle of $t_x = 6$ milli sec.

Process Y Suppose it has the CPU cycle of $t_y = 3$ milli sec.

Process Z Suppose it has the CPU cycle of $t_z = 2$ milli sec.

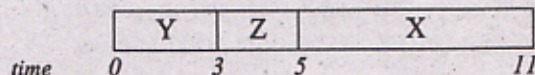
The operating system can manage these processes as follows:

Case 1: The total execution is as follows if three processes become ready for execution in the order of XYZ



$$T = (6 + 9 + 11) / 3 = 8.66 \text{ milli sec}$$

Case 2: The total execution is as follows if three processes become ready for execution in the order of YZX



$$T = (3 + 5 + 11) / 3 = 6.33 \text{ milli sec}$$

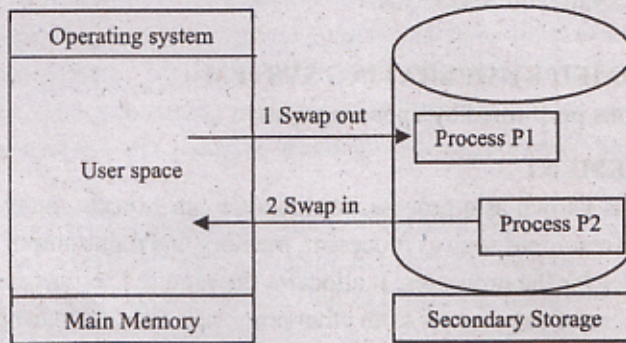
The operating system is managing the processes more efficiently in case 2.

2- MEMORY MANAGEMENT

Memory management is one of the important functions of operating system. It controls and manages the operation of main memory. It allocates the required memory for program and data.

It also de-allocates (free) memory when a program is closed and it is no more required. Then it updates the status of the memory.

Example: The following example shows the memory management for two processes P1 and P2. The process P2 is being swapped in (loaded) and P1 is being swapped out (taken out) of the memory.



3- FILE MANAGEMENT

File management is another important function of operating system in which operating system manages files and folders on storage devices such as hard disk, USB flash drive and DVD etc. It allows user to perform operations such as creating, deleting, copying, moving, renaming and searching files and folders. It also allows the user to perform read, write, open and close operations on files and folders.

4- I/O MANAGEMENT

I/O (input/output) management is a function of operating system in which operating system controls all the operations of input/output devices attached to the computer. The user communicates with the computer using different input/output devices such as keyboard, mouse and printer etc. The operating system manages the operation of these devices that improves the performance of computer.

Example: Suppose three programs, P1, P2 and P3 needs to use a printer. The operating system decides which program is allowed to use it first. It will create a queue and each program will use the printer turn by turn.

5- SECONDARY STORAGE MANAGEMENT

Secondary storage management is a function of operating system in which operating system manages the secondary storage i.e hard disk. The operating system manages the free space and data on the storage devices. It also allocates and de-allocates the storage space to user programs.

Example: Suppose a file F1 needs to be stored on the hard disk. The operating system will check the free space on the disk. It will assign a proper address for the file if space is available. Otherwise the operating system will show a message to empty the space.

6- NETWORK MANAGEMENT

Network management is a function of operating system in which operating system manages and monitors the resources of the network. It allows the user to create user groups and assign privileges to them. It allows sharing the network resources such as hard disk, printer, DVD etc to different users. It also detects and solves the network problems.

7- PROTECTION SYSTEM

Protection system is a part of operating system that ensures each resource of computer is used according to the privileges assigned to users by the system administrations. It protects the resources of a computer against unauthorized users. The users are provided specific username and password to prevent the misuse of the system.

8- COMMAND INTERPRETER

Protection system is a part of operating system that provides an interface between user and the computer system. It reads and executes the commands entered by user through keyboard. The command interpreter in Windows is cmd.exe.

PROCESS

Program in execution is known as process. For example a program is written in a programming language such as C or C++, then it is compiled and the compiler creates a binary code. The original code and binary code, both are programs. When we actually run the binary code, it becomes a process. The process is a part of program under execution that is controlled by operating system. A process uses various resources of computer such as CPU time, files, I/O devices, memory etc. A program becomes a process when it is executed and loaded into the memory.

VARIOUS STATES OF PROCESS

Process states describe the nature of current activity in a process. The state of a process changes as it executes. There are five states of a process as follows:

- | | |
|---------------------|---|
| 1- New: | New is the very first state of a process when it is created. |
| 2- Ready: | Ready is the state when a process is ready for execution but it is waiting for the allocation of CPU time by the operating system. |
| 3- Running: | Running is the state when a process is being executed by the processor. The operating system assigns CPU t a process for execution. |
| 4- Blocked/Waiting: | Blocked or waiting is the state when the process is waiting for some event to occur before it can continue execution. For example, it may be waiting for the completion of I/O operation. |
| 5- Terminated: | Terminated is the state when a process completes its execution and no longer exists in the memory. |

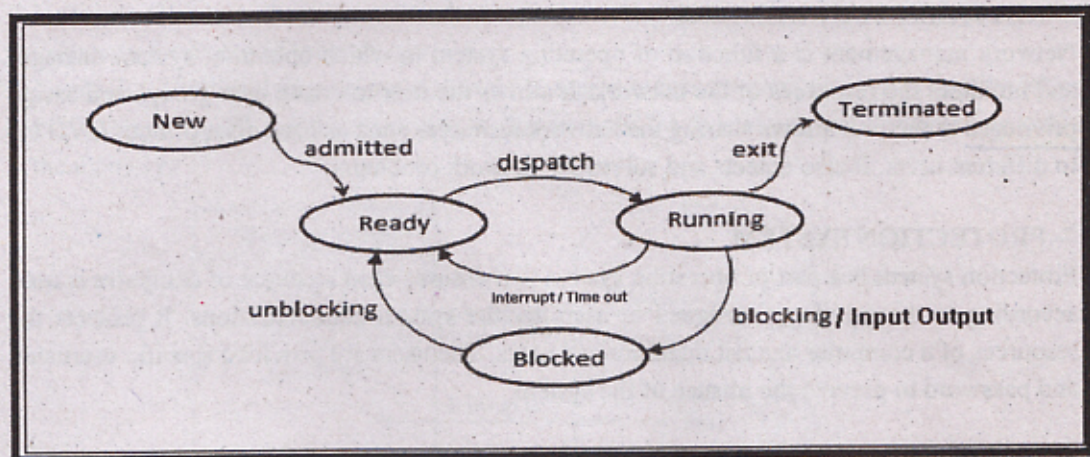


Diagram: States of Process

THREAD AND PROCESS

Processes and threads are the two basic units of execution. They execute a series of instructions. A process is a program in execution. A process may be made up of multiple threads. A thread is a sequence of instructions within a process that can be executed independently. The threads are made of and exist within a process; every process has at least one thread. A thread is also known as lightweight process.

DIFFERENCE BETWEEN PROCESS AND THREAD

PROCESS	THREAD
1- A process is a program in execution.	A thread is a subset of a process.
2- Process is a heavy weight entity.	Thread is a light weight entity.
3- Any change in process does not affect other processes.	Any change in a thread may affect the behavior of other threads.
4- Process runs in separate memory spaces.	Thread run in shared memory spaces.
5- It is controlled by operating system.	It is controlled by the programmer.
6- A process may continue running if a thread ends.	All threads end when the related process ends.

MUTLTI-THREADING

The process of executing multiple threads at the same time is known as multi-threading. It is an execution method of a program that allows a single process to execute different threads simultaneously.

Multi-threading allows multiple threads to exist within a single process. These threads can execute independently. The main purpose of multithreading is to maximize CPU utilization by executing different parts of a program at the same time. For example:

- A web browser can use one thread to display images and a second thread to retrieve data from the network.

- Web servers receive many requests for web pages, images and sounds etc. It uses multithreading to handle different requests.
- A user typing paragraph on MS word. But in the background one more thread is running and checking spelling mistakes.

MUTLTI-TASKING

Multi-tasking is the ability of an operating system to perform more than one task at the same time on the computer with single CPU. For example the user can edit a text document while a webpage is being loaded in the browser. It executes multiple tasks the same time by rapidly switching the CPU among them. The process of executing multiple threads at the same time is known as multi-threading. It is an execution method of a program that allows a single process to execute different threads simultaneously.

MUTLTI-PROGRAMMING

Multi programming is the ability of an operating system in which multiple programs are loaded in the memory and CPU executes one program at a time. The other program waits until previous program is executed or blocked for example, when a user loads two programs MS-word and language compiler. The CPU is able to execute only one program MS-word or language compiler at a time.

The advantage of multiprogramming is that it executes the programs quickly. The time is not wasted in loading the programs one by one. On the other hand the disadvantage is that it requires more amount of memory to load multiple programs at the same time. The large programs may not be fully loaded in memory and may run slowly.

MUTLTI-PROCESSING

Multi-processing is the ability of an operating system to execute more than one process simultaneously on computer with having multi processors. In this type of processing, two or more processors work together to execute the program at the same time. It also increases the processing speed of the computer.

EXERCISE

Q1. Select the best answer for the following MCQs.

1. In which operating system, same types of jobs are grouped together and executed one by one?
a) Multiprogramming operating system b) Batch processing operating system
c) Real-time operating system d) Time-sharing operating system
2. In _____ operating system CPU is rapidly switched between programs so that all the programs are executed at the same time.
a) Multiprogramming operating system b) Batch processing operating system
c) Real-time operating system d) Time-sharing operating system
3. Which operating system runs applications with very precise timing and provides immediate response to avoid safety hazards?

- a) Real-time operating system b) Multitasking operating system
c) Multiprocessing operating system d) Distributed operating system
4. _____ operating system divides a task into many subtasks and processes them independently using many processors.
a) Real-time operating system b) Distributed operating system
c) Parallel processing operating system d) Multitasking operating system
5. Which of the following manages allocation of computer resources during program execution?
a) Memory management b) Process management c) I/O management d) File management
6. Which of the following creates groups and assigns privileges to them?
a) Process management b) I/O management c) File management d) Network management
7. In which state, a process is waiting to be assigned to the processor by the operating system scheduler?
a) New state b) Ready state c) Waiting state d) Running state
8. In DOS, a user communicates with the operating system by issuing:
a) Commands b) instructions c) Routines d) Procedures
9. The interface used by Windows is called
a) Menu-driven interface b) Command line interface
c) Graphical user interface d) Prompt interface
10. GUI stands for?
a) Graphical User Interface b) General User Interface
c) Grayed User Interface d) Graphs, Utilities and Icons
11. _____ is the ability of an operating system to control the activities of multiple programs at the same time.
a) Multitasking b) Streamlining c) Single tasking d) Multithreading
12. The response time is very critical in _____ operating system
a) Time sharing b) Batch c) Quick response d) Real-time
13. The technique of moving the process in and out of memory is known as:
a) Connecting b) Swapping c) Switching d) Shifting
14. A program in execution is called:
a) Process b) Action c) Task d) Token
15. Which of the following is not a state of the process?
a) Running b) Ready c) New d) Privileged
16. Which of the following process state is known as waiting state?
a) Running state b) Ready state c) Blocked state d) Terminated state
17. A process is in _____ state when it is waiting for I/O to be completed.
a) Wait b) Ready c) Run d) Exit
18. Which of the following state transitions is not possible?
a) Blocked to running b) Ready to running c) Blocked to ready d) Running to blocked
19. Which of the following is also called as light weight process?
a) Function b) Module c) Kernel d) Thread
20. A process can be:
a) Single threaded b) Multithreaded c) Both a and b d) None

Q2. Write short answers of the following questions.

1. **What is the purpose of operating system in a computer?**

Ans. The main purpose of Operating System is to manage the computer resources such as a CPU, memory, disk drive etc. It controls the execution of application program and act as an interface between the user of a computer and the computer hardware. It loads programs into main memory and executes them. It also provides security through username and password.

2. **What is Graphical User Interface (GUI)?**

Ans. A type of user-interface that allows the user to give commands to computer through icons, menus, and buttons. The user does not need memorize commands like command line interface. It is very easy and user friendly interface. Today, it is the most commonly used interface of operating system all over the world.

3. **Mention three advantages of UNIX operating system.**

Ans. 1- UNIX supports multi user and multi-tasking.
2- It provides greater processing power and better security than windows.
3- The computer running UNIX OS usually do not get malware attack.

4. **Differentiate between multiprogramming and time-sharing operating systems.**

Ans. The main difference between multiprogramming and time-sharing OS is that multiprogramming system maximizes processor use, whereas the time-sharing system minimizes the response time. Time sharing system operates in an interactive mode with a quick response time. Multiprogramming does not support interactive computing. Another difference is that time-sharing OS gives each program a short period of CPU time known as time slice.

5. **Why multiprocessing operating systems have been developed?**

Ans. The multiprocessing operation system have been developed to execute a program by two or more processors at a time. It allows to process a large amount of data at very high speed. It improves the processing speed. Two or more CPUs in one computer can efficiently perform the task.

6. **Differentiate between single-user and multiuser operating system.**

Ans. The single-user OS allows only one user to use computer at a time. It is typically used on microcomputers. DOS is an example of single-user OS. The multi-user OS allows many users to use the single computer called server in a network. It is typically used on servers in business and offices where many users have to access the same application. UNIX, Linux, Windows 2000 onward and Mac OS X are examples of multi-user OS.

7. **Why memory management is required in a computer?**

Ans. Memory management is required in a computer to ensure that all the programs in memory are executed without any problem. It allocates space to programs that are loaded in main memory for execution. It also keeps track of de-allocated memory when a program is closed and updates the memory status.

8. **Why protection system is required in a computer?**

Ans. Protection system is required in a computer to protect different resources of a computer against unauthorized users and processes. It ensures that only authorized users can access the system. It also protects a process from the activities of other processes if multiple processes are

executing simultaneously. The users are assigned specific username and password to prevent the misuse of the system.

9. What is thread?

Ans. A thread is a sequence of instructions within a process. It is the basic unit of CPU allocation in modern operating systems. A process may have multiple threads in it.

10. Differentiate between multiprogramming and multithreading by giving one example of each.

Ans.

MULTIPROGRAMMING	MULTITHREADING
In multiprogramming, multiple programs are loaded in the memory and CPU executes on program at a time. The other programs wait until the first program is completed or blocked. For example, the operating system will execute the second program while the first program is waiting for the input from the user.	In multithreading, multiple threads execute at the same time. It maximizes CPU utilization by executing different parts of a program at the same time. For example, a web browser can use one thread to display images and a second thread to retrieve data from the network.

Q3. Write long answers of the following questions.

1. Mention the tasks performed by operating system.

Ans. See Ans on Page 6

2. Compare DOS with Windows operating system.

Ans. See Ans on Page 8

3. Describe the following types of operating system.

- Real-time operating system.
- Parallel processing operating system.
- Embedded operating system.

See Ans on Page 9,10

See Ans on Page 10

See Ans on Page 10

4. Define the following terms.

- File management
- I/O management
- Network management
- Command-interpreter

See Ans on Page 12

See Ans on Page 12

See Ans on Page 13

See Ans on Page 13

5. Describe the five states of process with diagram.

Ans. See Ans on Page 13, 14

UNIT 2

SYSTEM DEVELOPMENT LIFE CYCLE

SYSTEM

A system is a set of components such as hardware and software that are used to collect, create, store, process and distribute information. In today's life almost every person interacts with many systems during daily activities for example, an examination system determines the overall performance of each student, and a payroll system is used to manage employee's salaries etc.

SYSTEM DEVELOPMENT LIFE CYCLE (SDLC)

System development life cycle (SDLC) is a conceptual model used in project management that describes the stages involved in system development project. It is a framework that describes the activities performed at each stage of a software development period. The stages are known as phases of SDLC. SDLC is commonly used in software engineering to create or modify the information system.

IMPORTANCE OF SDLC

System development life cycle is important for the following reasons:

1. It is a systematic way of developing successful system.
2. It divides the work into different phases that makes the work easier.
3. It delivers the quality system according to the requirement.
4. It delivers cost-effective (low cost) system.
5. It provides effective ways to control and track the project.
6. It increases the development speed.

OBJECTIVES OF SDLC

1- QUALITY ENSURANCE

It is an important objective of system development life cycle used to ensure that high quality system is delivered. Each phase of SDLC makes sure that the system is developed according to the requirements.

2- MANAGEMENT CONTROL

Another objective of system development life cycle is to establish an appropriate level of management over the projects. The management control is very important to direct, coordinate, control, review and approve software development projects.

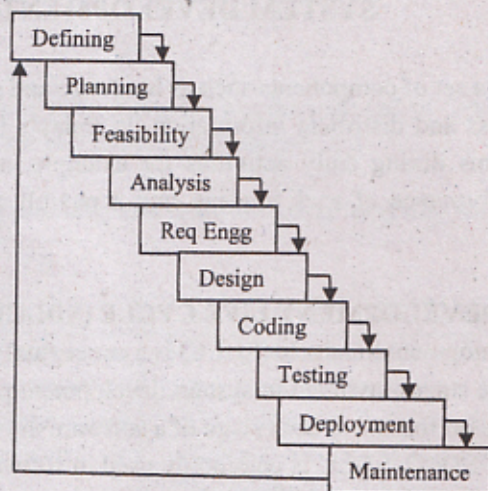
3- MAXIMIZE PRODUCTIVITY

It is another important objective of system development life cycle that is used to maximize productivity. SDLC identifies the potential risks in the project. It ensures that proper planning is done to avoid these risks that increase productivity.

PHASES OF SYSTEM DEVELOPMENT LIFE CYCLE (SDLC)

Following are the phases of system development life cycle:

1. Defining Problem
2. Planning
3. Feasibility
4. Analysis
5. Requirement Engineering
6. Design
7. Coding
8. Testing / Verification
9. Deployment / Implementation
10. Maintenance / Support



1- DEFINING PROBLEM

It is the first phase of SDLC in which the problem to be solved or system to be developed is clearly defined. All requirements related to the project are documented and approved from the customer or the company. The project is designed and developed according to these requirements.

Example: Student's Examination System Development.

Defining the problem: A student's examination system needs to be developed to cover all aspects from conducting examination and generating the results.

2- PLANNING PHASE

The planning phase of SDLC is used to determine the objective of the project. It considers the requirements for developing the project. It is also used to prepare an estimate of different resources such as personnel and cost. The information is analyzed to determine if any alternative solution is available to create the new product. If there is no other suitable alternative, the information is assembled into a project plan and presented to management for approval.

Example: The planning for a student examination system is used to set the goals. It also prepares an estimate of the resources such as personnel and cost.

3- FEASIBILITY

Feasibility study is the process of assessing the strength and weaknesses of the proposed system. It presents the directions of the activities that will improve a project and achieve required results. Feasibility study determines whether the project is technically, financially, legally and operationally feasible within the estimated cost and time.

TYPES OF FEASIBILITY STUDY

Feasibility study can be divided into five types as follows:

- **Technical Feasibility:** It is used to determine the level and type of technology needed for the system.
- **Economic Feasibility:** It is used to determine whether the system can be developed within the budget or not.
- **Operational Feasibility:** It is used to determine how well the proposed system will solve the problems.
- **Legal Feasibility:** It is used to determine the legal issues related to the system such as cyber law and copyright law etc.
- **Schedule Feasibility:** It is used to determine whether the system can be developed within the given time period or not.

Example: The student examination system will be assessed for all types of feasibilities. The reports will be presented to the management for final approval.

4- ANALYSIS PHASE

The analysis phase of SDLC is used to determine the requirement of the end-user. It is typically done with the help of client focus group, a group that interacts with the system user. The system users provide an explanation of their needs and expectations from the new system.

In this phase the project manager must decide whether the project should go ahead with the available resources or not. The project team asks the following questions during the analysis:

- Can proposed system be developed with available resources and budget?
- Will the new system significantly improve the organization?
- Does the existing system even need to be replaced?

Example: The student examination system is analyzed for the development. The team may visit the educational institute to study existing system to suggest improvements.

5- REQUIREMENT ENGINEERING

The requirement engineering is the process of determining user expectations from a new modified system. It is a set of activities used to identify and communicate the purpose of software system, and the framework in which it will be used. The requirement engineering consists of the following steps:

i. **Requirement gathering:** It is usually the first part of any software/system development process. It involves meetings with the customers. It is also used to analyze the market requirement and features that are in demand. These requirements are of two types:

- **Functional requirements:** The functional requirement specifies the software functionality. The developers must build the project according to these requirements to enable the users to accomplish their tasks.

- **Non-functional requirements:** The non-functional requirements specify the criteria for the judgment of the operations of a system. It describes how well the system performs its tasks.

ii. Requirement validation: The requirement validation is used to examine the requirements to ensure that they meet the intentions of the stakeholders. The validation differs from verification as the verification occurs after requirements have been accepted. The requirements validation process reviews requirements to check they are complete and accurate.

iii. Requirement management: The requirement management is performed to ensure that software continues to meet the expectations of the users. This process needs to gather new requirements that arise from changing expectations, new regulations or other changes.

Example: In student examination system, the development team will gather the required information by interviewing the users, using questionnaires or studying the existing documents.

6- DESIGN PHASE

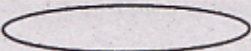
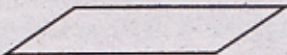
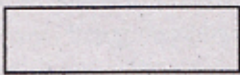

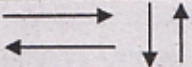

The design phase is the architectural phase of system design. In this phase the flow of data processing is developed into charts. The project team determines the most logical design and structure of data flow and storage. The project team also designs mock-up screen layout that the developers uses to write the code for the actual interface. The design phase normally consists of two different structures as follows:

i- Algorithm: An algorithm is a step-by-step procedure to solve a specific problem. The algorithm is are written before writing the actual computer program. An algorithm is written in simple English language to describe various operations. For example the terms Get, Read or Input are used to describe input operation. The term Print, Write or Output are used for the output.

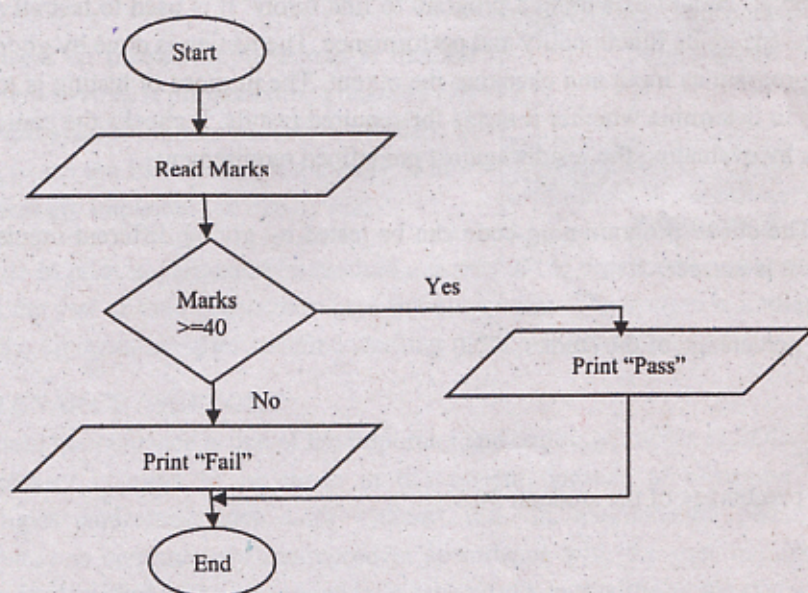
Example: In student examination system the following algorithm can be used to input the marks and decide if the student is pass or fail:

Step 1:	Start
Step 2:	Input marks
Step 3:	If marks ≥ 40 then print "Pass" else print "Fail"
Step 4:	End

ii- Flowchart: Flowchart is the pictorial representation of algorithm or program. It uses different symbols to show the steps of the algorithm. These symbols are linked together with arrows to show the flow of the process. The flow charts are used to analyze, design, document and manage a process or program. Following symbols are used in flowchart:

Symbol	Description
	It is called terminal symbol. It is used to represent Start and End in the flowchart.
	It is called parallelogram symbol. It is used to represent input and output in a flowchart.
	It is called rectangle symbol. It is used to represent process in a flowchart such as arithmetic operations.
	It is called diamond symbol. It is used to represent decision/condition in the flowchart.
	They are called flow lines. They are used to link one symbol to other symbol in flowchart.
	It is called connector symbol. It is used to join two kinds of algorithm.

Example: In student examination system the following flowchart for the above algorithm can be used to input the marks and decide if the student is pass or fail:



7- CODING / CONSTRUCTION PHASE

The coding / construction phase is used to execute the plan that is prepared in design phase. The programmers write the code for data flow process. They also write code for the design of actual user interface screens. The code is written in programming languages. Coding is also called computer programming.

Example: An example of coding written in C++ language is as follows:

```
#include<iostream>
#include<conio.h>
int main()
{
    int marks;
    cout<<"Enter marks: ";
    cin>>marks;
    if(marks>=40.0)
        cout<<"Pass";
    else
        cout<<"Fail"
    getch();
    return 0;
}
```

8- TESTING / VERIFICATION

Testing is the process of executing a program to find errors. It is used to test all aspects of the system to ensure its functionality and performance. The testing is done by giving sample data to the program as input and checking the output. The purpose of testing is to evaluate the program to determine whether it meets the required results. It checks the consistency of the program by evaluating the results against predefined requirements.

Example: The above programming code can be tested by giving different inputs to make sure the result is correct:

Test1:

Enter percentage of the student: 70.0

Pass

Test2:

Enter percentage of the student: 30.5

Fail

9- IMPLEMENTATION / DEPLOYMENT

Software deployment is a set of activities used to make the system available for use. The deployment is also known as implementation. A system can be implemented after it has been tested. The activities involved in deployment / implementation phase are as follows:

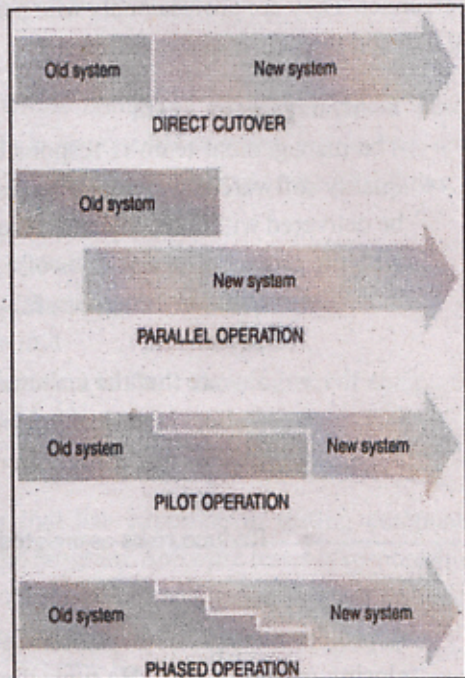
- Installation of hardware and software.
- The users and operation personnel are trained on developed software if required.

- **Conversion:** The process of changing from old system to the new one is called conversion. The conversion is done if it is required.

Methods of implementation / deployment

Following four methods / techniques are used for system deployment / implementation:

1. **Direct:** In direct implementation method the old system is completely dropped and the new system being completely implemented. It is most risky implementation because the old system is no longer available. It may be used when time is very short.
2. **Parallel:** In parallel implementation method both old system and new system is operated together for some period of time. It is safest but most expensive type of implementation. The results of both systems are compared, if the results of new system are satisfactory then the old system is stopped and new system is implemented otherwise the old system can be used until all problems in the new system are removed.
3. **Phased:** In phased implementation method the new system is implemented gradually and the old system is progressively discarded. In this phase the individual components of new system are implemented one by one.
4. **Pilot:** In pilot implementation method one part of the organization uses the new system and the rest of the organization uses the old system. The system is implemented in the whole organization after satisfied working of the new system.



9- MAINTENANCE / SUPPORT

The system maintenance is one of the important and ongoing process of SDLC. In this phase the necessary changes to be made in the system because of changing requirements, technologies and experience with system' use. In maintenance phase, the system performance is continuously monitored in accordance with the user requirements. If the problem is identified in the performance, it may results in modifying the system. The system may enter the planning phase if any modifications are required. This process continues until a complete solution is provided to the customer. The maintenance can be repairing, modification or enhancement in the system.

STAKEHOLDER

Stakeholder refers to any person or group who is directly or indirectly affected by the system. The stakeholder can either be within the organization or outside of the organization that

sponsor, plan, develop or use a project. It is the duty of the project management team to identify the stakeholders, determine their requirements, expectations to ensure a successful project. Examples of stakeholders are project manager, system analyst, programmer, software tester and users.

PERSONNEL / STAKEHOLDERS INVOLVED IN SDLC AND THEIR ROLE

SDLC's activities are performed by different groups of people and individuals that are called personnel. They are professionals who perform their particular jobs. The personnel involved in SDLC and their roles are as follows:

1- Management Team

The management team is responsible to satisfy the customers. It makes sure that high quality software is developed according to the customer's expectation. The project must be delivered within the allocated time and cost. The team also reduces the risks associated with the project. The key roles of a good management team is as follows:

- Provide a successful software with respect to time, cost and quality objectives.
- Ensure that the customer expectations are met.
- Collect historical information and data for future use.
- Ensure that all requirements are fulfilled through detailed work definition process.
- Reduce risks associated with the project.

2- Project Manager

A project manager is a professional that is responsible for planning, executing, and closing of any project. He typically has management skills with software development experience. He must also have the knowledge about system development life cycle. The key roles of a project manager are as follows:

- Developing the project plan.
- Managing the project schedule and budget.
- Managing the project stakeholders.
- Managing the project team.
- Managing the conflicts and risks.

3- System Analyst

A system analyst is a professional software developer. He studies the problems, plans solutions for problems, recommends software systems, and manages project development. A system analyst typically has the expertise in various programming languages, operating systems and hardware platforms. Some key responsibilities of a system analyst are as follows:

- Plans the system flow.
- Interacts with the customers to know the project requirements and then produce business requirements documents.
- Defines technical requirements.
- During system development phase he helps computer programmers.
- Manages system testing.
- Document, requirements and contribute to user manuals.

4- Programmer

A programmer is a technical person who writes computer programs in computer programming language to develop software. He is responsible to write, test, debug, and maintain the program code. Some key responsibilities of a programmer are as follows:

- Writes the program code.
- Updates and modifies the existing code if required.
- Tests the program to make sure that is giving the required results.
- Debugs the program if it has errors in it.
- Prepares graphs, tables and analytical data to show the progress of computer program.

5- Software tester

A software tester is a computer programmer that has expertise in testing computer programs. He uses different techniques to test the program. Software tester is responsible for understanding the requirements, creating test scenarios, and preparing test data and executing them. He prepares report about the problems in the program.

6- Customer

A customer is an individual or an organization that is a current or potential buyer or user of the software product. Customers are also called clients but the only difference between the customers and clients is that the customers purchase the software products and the clients purchase services. Customers are the real evaluators of a software product. They evaluate the software by using and identifying its merits and demerits.

EXERCISE

Q1. Select the best answer for the following MCQs.

1. The first step in the system development life cycle is:
a) Analysis b) Design c) Problem identification d) Development and documentation
2. The organized process or set of steps that needs to be followed to develop an information system is known as:
a) Analytical cycle b) Design cycle c) Program specifications d) System development life cycle
3. Enhancements, up gradations and bugs fixation are done during the _____ step in the SDLC
a) Maintenance and evaluation b) Problem identification
c) Design d) Development and documentation

4. The _____ determines whether the project should go forward.
a) Feasibility b) Problem identification c) System evaluation d) Program specification
5. _____ spend most of their time in the beginning stages of the SDLC, talking end users, gathering requirements, documenting systems and proposing solutions.
a) Project managers b) System analyst c) Network engineers d) Database engineers
6. The entities have positive or negative influence in the project completion are known as _____.
a) Stakeholders b) Stake supervisors c) Stake owners d) None of above
7. System maintenance is performed in response to:
a) System changes b) Hardware and software changes
c) User requests for additional features d) All of the above
8. What come after the design phase?
a) Implementation b) Analysis c) Coding d) Support
9. Which of the following is not a type of feasibility study?
a) Economic b) Social c) Operational d) Technical
10. An algorithm is a:
a) Computer program b) Programming language c) Sequence of steps d) Problem statement
11. The graphical representation of steps to solve a problem is known as:
a) Flowchart b) Hierarchical chart c) Decision table d) Data flow diagram
12. The decision symbol in a flowchart indicates:
a) Progress b) Condition c) Output d) Input
13. Coding is also called _____.
a) Evaluating b) Programming c) Installing d) Testing
14. The _____ phase occurs when the testing phase is completed and the new system is ready to replace the old one.
a) Development b) Design c) Implementations d) Maintenance
15. The process of training personnel to use the new systems is done during:
a) System analysis b) System design c) System development d) System implementation
16. The type of conversion in which individual components of the new system are used one by one is called:
a) Direct conversion b) Pilot conversion c) Phased conversion d) Parallel conversion
17. In which type of system conversion, the old system is directly replaced by the new system?
a) Parallel b) Direct c) Phased d) Pilot
18. _____ typically have many years of systems analysis or programming expertise and provide leadership in software development efforts.
a) Programmers b) Project managers c) Technical writers d) Customer
19. The person who will use the information system directly or will use the information produced by the system is called _____.
a) System analyst b) End-user c) Interface designer d) Top manager
20. The _____ is the person who writes programs in a programming language.
a) Programmer b) Technical writer c) Interface designer d) Top manager

Q2. Write short answers of the following questions.

1. What is a system?

Ans. See Ans on Page 19

2. Name different phases of SDLC?

Ans. See Ans on Page 20

3. What are the objectives of SDLC?

Ans. See Ans on Page 19

4. Give some activities of planning phase.

Ans. Some activities of planning phase include determining the objective of the project, preparing an estimate of different resources as well as determining if any alternative solution is available.

5. Differentiate between functional and non-functional requirements.

Ans. See Ans on Page 22

6. Design phase is considered as the "architectural" phase of SDLC. Give reasons.

Ans. The design phase is considered as the architectural phase of SDLC because it determines the most logical design and structure for data flow and storage. It is used to design the user interface layouts as well as it also includes preparing algorithms and flowcharts.

7. Explain flow chart symbols.

Ans. See Ans on Page 23

8. What is the purpose of Testing/verification phase of SDLC?

Ans. See Ans on Page 24

9. Give main activities of implementation phase.

Ans. The main activities of this phase consists of the installation of hardware, software and training of users.

Q3. Write long answers of the following questions.

1. Define software development life cycle (SDLC). What are its objectives?

Ans. See Ans on Page 19

2. What is system? Where exactly the testing activities begin in SDLC?

Ans. See Ans on Page 19, 24

3. Why software development life cycle is important for the development of software?

Ans. See Ans on Page 19

4. Who are stakeholders of SDLC? Describe their responsibilities.

Ans. See Ans on Page 25

5. Write the purpose of Requirement Engineering phase? Explain its various steps in detail.

Ans. See Ans on Page 21

6. Which personnel are involved in SDLC? Explain their role briefly.

Ans. See Ans on Page 26

UNIT 3

OBJECT ORIENTED PROGRAMMING IN C++

COMPUTER PROGRAM

A computer program is a set of instructions that performs a specific task when executed by a computer. It tells the computer what to do and how to do. All the tasks performed by the computer are controlled by computer program. For example, Microsoft Word is a program that is used to create documents, similarly Skype is a program used to make calls to other people who are on Skype. The computer programs are written in high level programming languages. A program written in high level languages are not understandable by the computer therefore a software called compiler is used to translate the high level language program into machine language before execution.

Source code: A program written in a high level language by the programmer is called source code or source file.

Object code: A code that is generated by the compiler from source code is called object code. It is also known as machine code.

C++

C++ is a general purpose programming language. It was developed by Bjarne Stroustrup in early 1980s at bell laboratories. Its main purpose was to make writing good programs easily. It is an object oriented programming language that supports working with objects and classes. It is commonly used to develop various types of programs such as commercial software, games, and graphics programs. It is a case sensitive language therefore the instructions written in C++ language must be in lower case letters (small letters).

INTEGRATED DEVELOPMENT ENVIRONMENT

Most of the new programming languages use Integrated Development Environment to create, compile and run programs. IDE is actually a software that brings all the tools required for efficient program development into one place. C/C++ language IDE consists of the following main components:

- | | |
|---------------------|---|
| Text Editor: | Text editor is a simple word-processor that is used to create and edit source code of a program. |
| Compiler: | A compiler is a computer software that translates C/C++ language source program into machine program (object program). It also detects errors in the program and gives hints to the programmer to correct them. |
| Linker: | Linker is a software that combines the main function in object file with various other functions making single resulting executable file. |

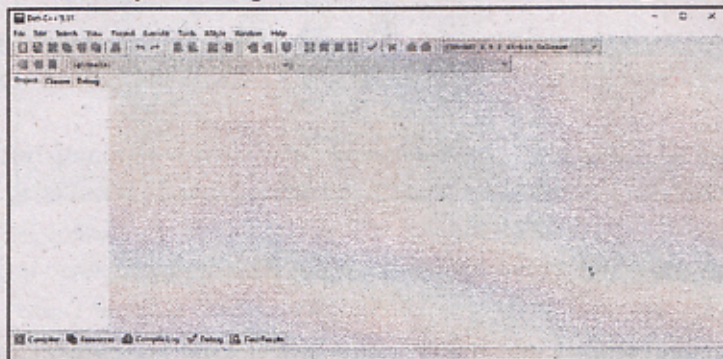
Different IDE's can be used to make C/C++ programs such as Borland C++, Turbo C++, and Dev C++ etc. In this book Dev C++ IDE is used for making programs.

Dev IDE

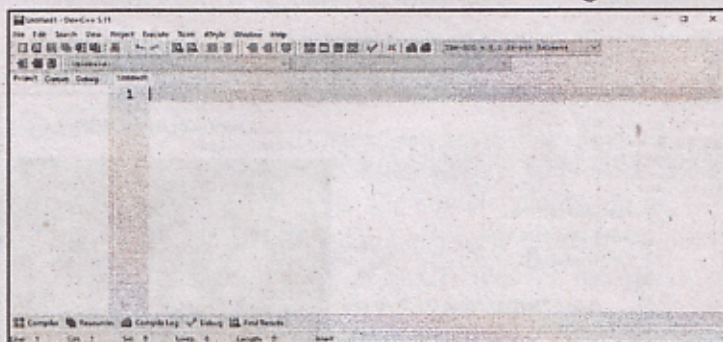
Dev C++, developed by Bloodshed Software, is a fully featured graphical IDE (Integrated Development Environment), which is enable us to create Windows-based C/C++ programs.

STEPS OF CREATING C++ PROGRAM IN DEV C++.

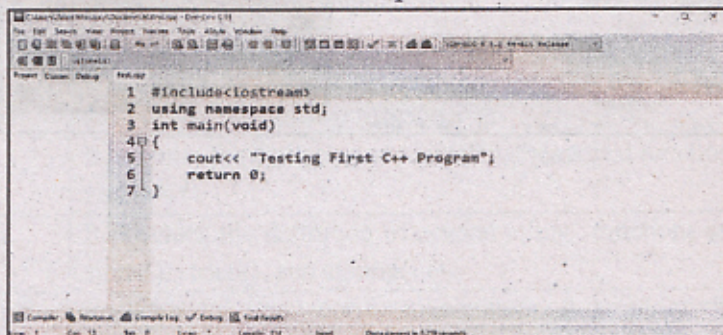
1- Open Dev C++ from your computer. You can see the following screen.



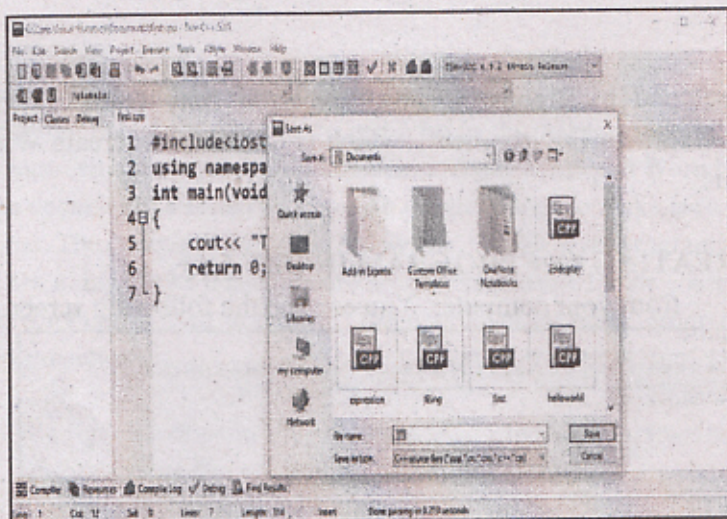
2- Click File→New→Source File. You will see the following screen.



3- Now write down the source code for example:

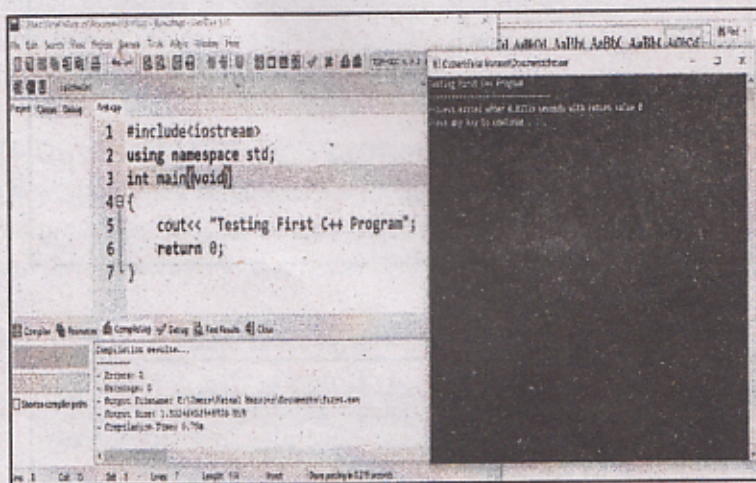


4- Now save the program by clicking File→Save.



Write File Name and select “C++ source files” for Save as type:

5- Now Compile and Run (Link) the program by Clicking Execute→Compile & run. It will show errors if there are errors in the program otherwise it will show the output of the program.



BASIC STRUCTURE OF C++ PROGRAM

The format of writing program is called its structure. A C++ program consists of the following main parts:

1- Preprocessor directive: The preprocessor directive is used to include a header file. It is used at the beginning of the program. The syntax of preprocessor directive to include header file in a program is as follows:

```
#include<name of header file>
```

For example: `#include<iostream>` // do NOT use .h with iostream if you are using dev.

OR `#include<iostream.h>` // USE .h with iostream if you are using turbo C++ or Borland C++.

Basic Structure of C++ program.

```
#include<headerfile.h>
main(void)
{
    Body of main function
}
```

2- main() function: The main() function is the starting point of a C++ program. When the program is executed, the control enters main() function and start executing its statements. Each program must contain a main() function. The location of the main() function is not important, if the program contains many functions. The main function is always executed first. A program that has no main() function cannot be executed.

3-Body of main() function: The C++ statements written between curly brackets(braces) { } are known as body of main() function. It may have any numbers of statements. These statements are actually the instructions for the computer to perform specific task. Each statement in C++ is terminated by a semicolon (;).

HEADER FILES

Header files contain information that is required by the program. C++ provides many header files for different purposes. Each header file contains definition of different functions and objects. The header file must be included in the program in order to use its functions. Header files have .h extension. Some examples of header files are iostream.h, conio.h, string.h and math.h etc.

The commonly used header files are listed in the following table with their purpose.

Header File	Purpose
iostream.h	It stands for input output stream. It contains the definition of basic input/output statements like cout and cin
conio.h	It stands for console input output. It contains the definition of function such as getch(), getche(), clrscr().
math.h	It provides the definition of several mathematical functions such as sqrt() and pow()
string.h	It provides the definition of several string functions such as strcpy(), strcat(), strlen(), and strcmp() etc.
iomanip.h	It stands for input output manipulator. It provides the definition of different manipulators like setw(), setprecision() etc.

RESERVED WORDS

Reserved words are special words that has predefined meanings. They are reserved by a developer of a programming language for specific purpose in programs. All reserved words are written in lower case (small) letters. These words cannot be used as variable name. Reserved words are also called keywords. There are about 80 reserved words in C++ depending on the version being used. The list of keywords in C++ is as follows:

and	and_eq	asm	auto	bitand
bitor	bool	break	case	catch
char	class	const	const_cast	continue
default	delete	do	double	dynamic_cast
else	enum	explicit	export	extern
false	float	for	friend	goto
if	inline	int	long	mutable
namespace	new	not	not_eq	operator
or	or_eq	private	protected	public
register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	struct	switch
template	this	throw	true	try
typedef	typeid	typename	union	unsigned
using	virtual	void	volatile	wchar_t
while	xor	xor_eq		

CONSTANT

Constants are the quantities whose values do not change during the execution of a program. It has of two types:

1- NUMERIC CONSTANT

Numeric constants are those constants which can be mathematically operated i.e. addition, subtraction, multiplication etc. can be performed on this type of constant. It also consists of two types:

i) **Integer constant:** Integer constants are the constants which do not have a decimal point in it. It represents values that are counted, like the number of students in a class. Integer constant may have negative or positive nature but cannot include commas or decimal point in it. e.g. 456.78 is an invalid integer constant because integer does not have decimal point in it. Similarly 4,325 is also invalid integer constant because integer does not have comma in it.

ii) **Floating-point constant:** Floating point numeric constants are the constants which include decimal point in it. It represents values that are measured like the height of a person etc.

2- CHARACTER CONSTANT

Character constants are the constants that cannot be mathematically manipulated. It is further sub-divided into two types:

i) **Character constant:** A type of data having a single length and must be enclosed within single inverted commas. Character constants can include digits from 0-9, upper case letters from A-Z, lower case letters from a-z, punctuation symbols such as semicolon, comma etc. and special symbols such as +, -, >, < etc.

ii) **String constant:** Sequence or combination of characters is said to be a string. It's a type of data having variable length and must be enclosed with double inverted commas. Fields like name, father's name, address etc. is stored in string.

VARIABLE

A variable is a memory space identified by a name, used to store certain kind of data. Each variable in C has a specific type, which determines the size, layout of the variable's memory, and the range of values that can be stored within that memory space.

RULES FOR NAMING A VARIABLE / VARIABLE NAME CONVENTIONS

- 1- A variable must begin with a character letter or underscore (_). Remaining characters may consist of digits, characters, underscore etc.
- 2- Only underscore can be used as a special symbol to improve readability of variable.
- 3- Blank space or comma is not allowed.
- 4- Both upper case and lower case letters are allowed.
- 5- The key words or reserved words such as if, for, void, and int etc. are not allowed.

VARIABLE DECLARATION IN C++

C++ is a strongly typed language because all variables must be declared before we use them in C program. Declaring variable tells the computer the name of the variable and its data type. This enables the compiler to recognize the valid variable name in your program.

data type variable_list;

Here, type must be a valid C++ data type including char, int, float, double etc, and variable_list may consist of one or more variable names separated by commas. Some valid variable declarations along with their definition are shown here:

```
int i, j, k;
char c, ch;
float f, salary;
```

VARIABLE INITIALIZATION IN C++

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is:

variable_name = value;

Variables can be initialized (assigned an initial value) in their declaration as follows:

type variable_name = value;

Some examples are:

```
int d = 3, f = 5;           // initializing d and f.
double pi = 3.14159;       // the variable pi will store the value 3.14159.
char x = 'x';              // the variable x has the value 'x'.
```

DATA-TYPES / TYPES OF VARIABLE IN C++

C++ provides three types of variable. OR There are three basic data type in C++ language.

1- Integer

Integer variable or data type is used to store numbers without decimal points i.e. the whole number. It may store positive or negative value. Depending upon the maximum and minimum values, following are the integer data types in C:

<u>Integer type</u>	<u>No. of bytes</u>	<u>Range</u>
int	4 bytes	-2147483648 to 2147483647
unsigned int	4 bytes	0 to 4294967295
short int	2 bytes	-32768 to 32767
unsigned short int	2 bytes	0 to 65535
long int	4 bytes	-2147483648 to 2147483647
unsigned long int	4 bytes	0 to 4294967295

2- Float / Real Number

Float data type is used to store numbers with decimal point i.e real numbers. It can store positive as well as negative real numbers. Depending upon the maximum and minimum values, following are the types of float data types in C.

<u>Float type</u>	<u>No. of bytes</u>	<u>Range</u>
float	4 bytes	-3.4^{38} to 3.4^{38} (with 6 digits of precision)
double	8 bytes	-1.7^{308} to 1.7^{308} (with 15 digits of precision)
long double	10 bytes	-1.7^{4932} to 1.7^{4932}

3- Character

Character variable or data type is used to store single character. One byte of memory is set aside to store a single character.

THE const QUALIFIER

The const qualifier in C++ is used to define a variable whose values cannot be changed throughout the program. The variable defined with the const qualifier must be assigned some value. Normally the name of this type of variable is written in uppercase letters.

Syntax for defining const qualifier:

```
const data_type variable=constant;
const int AGE=34;
const float LENGTH=7.5;
```

INPUT/OUTPUT HANDLING

In programming, providing data to the program is known as input. The input is normally given by the keyboard. The term standard input refers to the input through the keyboard. Similarly output means to display some data on the screen. The output is normally displayed on monitor. The term standard output refers to the output displayed on monitor. We can also save the data in a file on a storage medium.

In C++, input / output (I/O) is performed by using streams. A stream can be defined as flow of data. In C++ it is an object that is used in programs to insert or extract characters. The `cout` object is used for standard output to display data on screen and `cin` object is used for standard input to get data from keyboard. The header file `iostream.h` must be included in the program in order to use `cin` and `cout` objects in the program.

The cout statement

The `cout` statement is used to display text or values on the screen. `cout` stands for console output. The syntax of `cout` statement is as follows:

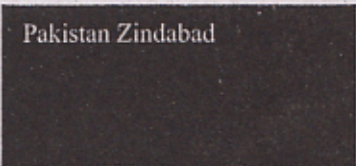
```
cout<< "string" / variable;
```

<code>cout</code>	It is output statement
<code><<</code>	It is known as insertion operator that sends the output to <code>cout</code> object. The <code>cout</code> object then sends the output to the screen. We can use multiple insertion operator in one <code>cout</code> statement.
<code>string / variable</code>	It refers to the string/message or the value of variable/constant that would be displayed on the screen.

Example 1:

```
#include<iostream> // use iostream.h if you are using turbo or borland C++ IDE.
using namespace std; // do NOT this statement if you are using turbo or borland C++ IDE.
int main(void)
{
    cout<< "Pakistan Zindabad";
    return 0;
}
```

Output:



Pakistan Zindabad

Example 2:

```
#include<iostream>
using namespace std;
int main(void)
{
    int a; a=10;
    cout<< a;
    return 0;
}
```

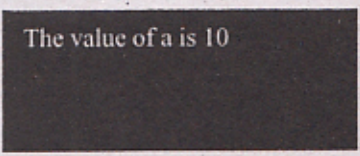
Output:



10

Example 3:

```
#include<iostream>
using namespace std;
int main(void)
{
    int a; a=10;
    cout<< "The value of a is"<<a;
    return 0;
}
```

Output:


The value of a is 10

ESCAPE SEQUENCE

Escape sequences are the special characters used to control the format of output. These characters are used in output statement (cout) to specify certain actions such as new line or to print certain characters like "" (double quotes). The escape sequence begins with back slash (\) followed by the character. The back slash (\) indicates that a special effect is needed and the character following the back slash specifies what to do. Following are the commonly used escape sequences used in c-language:

Escape Sequence	Purpose
\a	"a" stands for alert or alarm. This character causes a beep sound in the computer internal speaker.
\b	"b" stands for Backspace. This escape sequence causes the cursor to move one space back.
\f	"f" stands for Form Feed. This escape sequence leaves one blank paper on the printer attached with computer while printing.
\n	"n" stands for New line. It moves the cursor to the beginning of next line.
\r	It is used for carriage return. . It moves the cursor to the beginning of the line.
\t	"t" stands for tab. This escape sequence causes the cursor to move one tab forward.
\\	There are two backslash characters. One is used as control character and second is used to print a back slash '\ ' character.
\'	This escape sequence is used to print a single quotation mark
\"	This escape sequence is used to print Double quotation mark.

Example program of the usage of escape sequences is as follows:

```
#include<iostream>
using namespace std;
int main(void)
{
    cout<< "Pakistan\n Zindabad\n";
}
```

```
cout<< "Pakistan\t Zindabad\n";
cout<< " \"Pakistan Zindabad\" ";
cout<< "\n Pakistan\\Zindabad";
return 0;
}
```

Output:

```
Pakistan
Zindabad
Pakistan      Zindabad
"Pakistan Zindabad"
"Pakistan\\Zindabad"
```

THE MANIPULATORS

C++ manipulators are the commands that are used to format the output in different ways. There are many manipulators in C++. The most commonly used manipulators are **endl** and **setw**.

i- endl manipulator

The endl manipulator is used to move the cursor to the beginning of next line. It works similar to '\n' escape sequence. The endl manipulator is used by including the header file iostream in the program.

Example program of the usage of endl manipulator is as follows:

```
#include<iostream>
int main(void)
{
cout<< "Pakistan"<<endl;
cout<< "Zindabad";
cout<<endl<< " It's a beautiful country";
return 0;
}
```

Output:

```
Pakistan
Zindabad
It's a beautiful country
```

ii- setw manipulator

The setw manipulator is used to display a value or string in specified columns. The setw manipulator is used by including the header file iomanip.h the program. The syntax of setw is as follows:

setw(n)

Here n indicates the number of columns in which the value is to be displayed. The output will be right justified.

Example program of the usage of setw manipulator is as follows:

```
#include<iostream>
#include<iomanip>
using namespace std;
int main(void)
{
int prin=100000,prof=85000,lect=70000;
cout<<"Designation "<<setw(10)<<"Salary"<<endl;
cout<<"Principal "<<setw(10)<<prin<<endl;
```

Output:

Designation	Salary
Principal	100000
Professor	85000
Lecturer	70000

```

cout<<"Professor" <<setw(10)<<prof<<endl;
cout<<"Lecturer" <<setw(10)<<lect<<endl;
return 0;
}

```

The cin statement

The cin statement is used to get data from the keyboard and assign it to one or more variable. The cin stands for console input. The syntax of cin statement is as follows:

```
cout<<variable;
```

cin

It is input statement used to get data from keyboard.

>>

It is known as extraction operator that gets the input from cin object. The cin object then stores the input to variable. We can use multiple extraction operators in one cin statement.

variable

It refers to the variable in which the input value is stored.

Example1:

```

#include<iostream>
using namespace std;
int main(void)
{
    int a;
    cout<< "Enter the value of variable a";
    cin>> a;
    cout<< "The value of variable a is"<<a;
    return 0;
}

```

Output:

Enter the value of variable a 15
The value of variable a is 15

Example 2:

```

#include<iostream>
using namespace std;
int main(void)
{
    int a,b;
    cout<< "Enter two values";
    cin>>a>>b;
    cout<< "a="<<a<< "and"<< "b="<<b;
    return 0;
}

```

Output:

Enter two values 20 70
a=20 and b=70

Note: Whenever the words "get", "input", "accept", "read", "take" come in the question then we have to use cin statement in the program.

EXPRESSION

Expression is a combination of operator and operand. The operand may be a constant or variable. For example $10 + 20$ is an expression in which $+$ is an operator, whereas 10 and 20 are operands.

OPERATOR

An operator is a symbol that tells the compiler to perform specific operation like mathematical operation, assignment operation, logical operation etc. C language is rich in built-in operators and provides following type of operator

- | | | |
|------------------------|------------------------|-------------------------------------|
| 1- Assignment operator | 2- Arithmetic operator | 3- Arithmetic Assignment operator |
| 4- Relational operator | 5- Logical operator | 6- Increment and Decrement operator |
| | 7- Ternary operator | |

1- ASSIGNMENT OPERATOR (=)

= sign is used as an assignment operator in C-language. It is used to assign a value to a variable. For example $z = 10$, here 10 is a value which is assigned to a variable z.

The value that can be written on the left hand side of assignment operator is known as Lvalue and the value that can be written on the right hand side of assignment operator is known as Rvalue. Here it should be noticed that Lvalue can never be a constant value and can never be an expression for example:

$10 = a$ is a wrong statement because Lvalue is a constant. It can be written as $a = 10$. Similarly $b + c = a$ is a wrong statement because Lvalue is an expression/formula. It should be written as $a = b + c$.

2- ARITHMETIC OPERATOR

Arithmetic operators are used to perform arithmetic operations such as addition, subtraction, multiplication etc. Following table shows all the arithmetic operators supported by C++ language. Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Used for Addition.	$A + B$ will give 30
-	Used for Subtraction.	$A - B$ will give -10
*	Used for Multiplication.	$A * B$ will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	$B \% A$ will give 0

COMPOUND ARITHMETIC EXPRESSION

An expression which comprises of different arithmetic operators is said to be a compound arithmetic expression e.g $6 + 6 * 4 \% 2$

PRECEDENCE OF ARITHMETIC OPERATORS

Precedence determines which operator is used first in a compound expression. There are two precedence levels for arithmetic operators that are high and low. The high priority operators

are *, /, % and low priority operators are + and -. When a compound expression is evaluate, all high-priority operators are performed first in order in which they appeared and then all low-priority operators are carried out, if there is bracket in the expression then the expression inside the bracket is evaluated first.

Example 1:	$90 - 17 / 2 * 10 + 5 \% 2$	1 st	$17 / 2$ is solved
	$90 - 8 * 10 + 5 \% 2$	2 nd	$8 * 10$ is solved
	$90 - 80 + 5 \% 2$	3 rd	$5 \% 2$ is solved
	$90 - 80 + 1$	4 th	$90 - 80$ is solved
	$10 + 1 = 11$ Answer		

Example 2:	$(10.0 + 15.0) / (2 + 4.3)$	1 st	$10.0 + 15.0$ is solved
	$25.0 / (2 + 4.3)$	2 nd	$2 + 4.3$ is solved
	$25.0 / 6.3 = 3.96825$ Answer		

Here it should be noted that the result of integer division will always be an integer. For example the result of $17/2$ will be 8 not 8.5. If any of the operand is float then the result would be float for example the result of $17.0/2$ will be 8.5.

Example program to understand all the arithmetic operators available in C++ programming language:

```
#include <iostream>    // use iostream.h if you are using turbo or borland C++ IDE.
using namespace std;  // do NOT this statement if you are using turbo or borland C++ IDE.
int main(void)
{
    int a = 21, b = 10;
    cout << "\n Addition of variable" << a << " and " << b << " is " << a + b;
    cout << "\n Subtraction of variable" << a << " and " << b << " is " << a - b;
    cout << "\n Multiplication of variable" << a << " and " << b << " is " << a * b;
    cout << "\n Division of variable" << a << " and " << b << " is " << a / b;
    cout << "\n Modulus of variable" << a << " and " << b << " is " << a % b;
    return 0; }
```

3- ARITHMETIC ASSIGNMENT OPERATOR

Arithmetic Assignment operator is the combination of arithmetic operator and assignment operator. They are also called compound assignment operator. Following operators are used as arithmetic assignment operator:

+=	is used for addition assignment
-=	is used for subtraction assignment
*=	is used for multiplication assignment
/=	is used for division assignment
%=	is used for remainder assignment

The above operators are used whenever the same variable appears on both side of the operator for example:

$a=a+10$	can be is written as	$a+=10$
$num=num/100$	can be written as	$num/=100$
$x=x-y$	can be written as	$x-=y$

4- RELATIONAL OPERATOR

Relational operators are used to compare two values of same type. These operators are also known as comparison operators. When they are used in an expression they produce result as true or false. (e.g. $4 > 3$ can be read as 4 is greater than 3, that produces true or false). Following relational operators are used in C++:

Operator	Read as
$>$	Greater than
$<$	Less than
$=$	Equal to
$>=$	Greater than or equal to
$<=$	Less than or equal to
$!=$	Not equal to

Example:

Suppose the value of $X=10$ and the value of $Y=5$, then different relational expressions will be evaluated according to the given table:

Relational Expression	Result
$X > Y$	True
$X < Y$	False
$X <= Y$	False
$X >= Y$	True
$X = Y$	False
$X != Y$	True

5- LOGICAL OPERATORS

These operators are used for forming compound conditions. In other words these operators are used to combine two or more conditions. Logical operators are

i) AND (&&)

It is a type of logical operator. $\&\&$ sign is used for AND. It is used to combine two relational expressions OR we can say that it is used for forming compound conditions. If both conditions are true then it returns true. If any of the condition is false then it returns false for example:

$(10 < 15) \&\& (5 = 5)$	will return	True(1)
$(10 < 15) \&\& (5 > 5)$	will return	False(0)
$(12 != 120) \&\& (91 < 91)$	will return	False(0)

ii) OR (||)

It is a type of logical operator. || sign is used for **OR**. It is used to combine two relation expressions OR we can say that it used for forming compound conditions. If both conditions are false then it returns false. If any of the condition is true then it returns true for example:

(10<15) (5==5)	will return	True(1)
(10<15) (5 > 5)	will return	True(1)
(12!=120) (91<91)	will return	True(1)

iii) NOT (!)

It is a type of logical operator. ! sign is used for **NOT**. It is used to inverse the result. If the result is true it inverses into false, and if the result is false then it inverses into true. For example:

! ((10<15) && (5==5))	will return	False(0)
! ((10<15) && (5 > 5))	will return	True(1)
! ((12!=120) && (91<91))	will return	True(1)

Precedence of Logical Operators

In an expression containing several logical operators, then the expression is evaluated according to precedence. In C++ NOT(!) has the highest priority, AND(&&) has the next highest priority and OR (||) has the lowest priority.

6- INCREMENT (++) AND DECREMENT (--) OPERATOR

The increment operator is represented by ++ and used to add 1 to the previous value of variable. For example if a=10 and we write a++ then the value of variable 'a' in memory becomes 11. The decrement operator is represented by -- and used to subtract 1 to the previous value of variable, for example if n=10 and we write n-- then the value of variable 'n' in memory becomes 9.

Postfix and Prefix Increment Operator

When the increment operator (++) appears after the variable like n++ then it is said to be a postfix increment operator. In postfix increment operator the increment occurs after the variable's value is used in the expression for example if b=10 and we write a=b++; then first the value of variable 'b' which is 10 is assigned to variable 'a' then the increment occurs in the variable 'b' and the value of 'b' becomes 11.

When the increment operator (++) appears before the variable like ++n then it is said to be a prefix increment operator. In prefix increment operator the increment occurs before the variable's value is used in the expression, for example if b=10 and we write a=++b; then first the increment occurs in value of variable 'b' and the value of variable 'b' becomes 11 and then this value is assigned to variable 'a'.

Postfix and Prefix Decrement Operator

When the decrement operator (`--`) appears after the variable like `n--` then it is said to be a postfix decrement operator. In postfix decrement operator the decrement occurs after the variable's value is used in the expression for example: If `b=10` and we write `a=b--`; then first the value of variable 'b' which is 10 is assigned to variable 'a' then the decrement occurs in the variable 'b' and the value of 'b' becomes 9.

When the decrement operator (`--`) appears before the variable like `--n` then it is said to be a prefix decrement operator. In prefix decrement operator the decrement occurs before the variable's value is used in the expression, for example: if `b=10` and we write `a=--b`; then first the decrement occurs in value of variable 'b' and the value of variable 'b' becomes 9 and then this value is assigned to variable 'a'.

Example program to understand increment and decrement operators available in C++ programming language:

```
#include <iostream>
using namespace std;
int main(void)
{
    int a = 21, b, c;
    b=a++;      //first the value of variable 'a' is assigned to variable 'b' and then increment in 'a' occurs.
    c=++b;      //first the value of variable 'b' is incremented 'b' and then it is assigned to variable 'c'.
    cout<<c--<<endl;    //first the previous value is displayed and then the value of c is decremented.
    cout<<b++<<endl;    //first the previous value of b is displayed and then the value of b is incremented.
    cout<<--a<<endl;    //first the the previous value of a is decremented and then the value of a is displayed.
    return 0;
}
```

7- TERNARY OPERATOR / CONDITIONAL OPERATOR

The `?:` operator is known as ternary operator or conditional operator. It evaluates a condition and returns one of two values based on the result of condition. It is a simple decision making structure. It is very useful when one of two options is selected on the basis of a given condition.

The general form/syntax of ternary operator:

condition ? expression1: expression 2;

condition	The condition is specified as relational or logical expression.
expression 1	It is executed if the condition is true.
expression 2	It is executed if the condition is false.

Example program to understand ternary operator available in C++ programming language.

```

#include <iostream>
using namespace std;
int main(void)
{
    int a,b,max;
    cout<< "Enter the first value of a and b";
    cin>>a>>b;
    max=(a>b)? a:b;
    cout<< "The maximum value is"<<max;
    return 0;
}

```

TYPE CASTING

The process of converting the data type of a value from one type to another is known as type casting. Type casting can be performed in two ways:

1- Implicit type casting

2- Explicit type casting

1- Implicit Type Casting

Implicit type casting is a type of casting that is performed by compiler. It automatically converts one datatype to another. For example variable x is declared as float and the following calculation is to be performed: $x=15/6$

When this integer division is performed, the result will also become an integer value 2 which will be implicitly converted to floating-point value 2.0 and assigned to the variable x because the data type of variable x is float.

2- Explicit Type Casting

Explicit type casting is performed by the programmer. It is performed by using special operator known as cast operator. The cast operator tells the compiler to convert the data type of a value. The syntax of using cast operator is as follows:

(type) expression;

type indicates the data type to which operand is to be converted and expression may be a constant, variable or expression whose data type is to be converted.

Suppose, x and y are variables of type int and z is of type float. The integer value 15 is stored in x and 6 in y and the following division is to be performed.

$z = x/y;$

When this division is performed, integer math will be used and the result produced will be 2 which will be assigned to the variable z

To obtain correct result, type casting should be used to convert at least one of the operands to type float as shown below.

$z = (\text{float})x/y;$

Now, first the value stored in x will be converted to type float and then the division will be performed. The floating-point math will be used and the correct result 2.5 will be produced

which will be assigned to z.

Similarly, the int type can also be used to convert a floating-point value stored in a floating point variable into integer type by truncating the fractional part of the number.

UNARY, BINARY AND TERNARY OPERATOR

The difference between unary, binary and ternary operators is as follows:

i- **Unary operator:** A type of operator that works with single operand is called unary operator. Some of the unary operators are as follows:

--, ++, -, !.

ii- **Binary operator:** A type of operator that works with two operands is called binary operator. Some of the unary operators are as follows:

+, -, *, >, < etc.

iii- **Ternary operator:** A type of operator that works with three operands is called unary operator. The three operands include a condition and two expressions. It evaluates a condition and executes one of two expressions. The symbol ?: is used as ternary operator.

getch() function

The word getch stands for get character. It is an input function that is used to get a single character from the user. When this function is executed, it waits for any key to be pressed. The character entered by the user is not displayed on the screen. It is useful when a program needs to read a character the instant it is typed without waiting for enter key. For example a game moves an object when the user presses an arrow key. It should not be required to press enter key after pressing the array key. The getch() function is defined in the conio.h library file.

getche() function

It is an input function that is used to get a single character the instant it is typed without waiting for enter key. The "e" in the getche() stands for echo on, which means the entered character will be displayed on the screen as well. It is defined in the conio.h library file.

```
#include <iostream>
#include <conio.h>      // used for getch() function
using namespace std;
int main(void)
{
    char ch1, ch2;
    cout << "Enter first character: ";
    ch = getch();      // without using 'e'
    cout << "Enter second character: ";
    ch = getche();     // with using 'e'
    cout << "The first character is:" << ch1;
    cout << endl << "The second character is:" << ch2;

    return 0;
}
```

Output:

```
Enter first character:
Enter second character: z
The first character is: a
The second character is: z
```

COMMENTS

Comments are the useful explanatory statements in the source code that can be read by humans but invisible to the compiler. Comments are non-executable statements that are usually used to explain the program logic and can be inserted anywhere in the program. There are two types of comments as follows:

i) Single line comment

Single line comments begin with `//` and are inserted in the beginning of the line. Anything written on the right side of double slash is considered as comments and ignored during execution. For example:

```
// Program to find the area of rectangle
```

ii) Multi line comment

The Multi line comments are useful to make more than one lines as comments. It starts with `/*` and ends with `*/`. For example:

```
/* Program to find the area of rectangle
```

The formula is:

```
A=l*w; */
```

EXERCISE

Q1. Select the best answer for the following MCQs.

- Which of the following is ignored during program execution?
a) Reserved word b) Constant c) Comment ✓ d) const qualifier
- What is the range of unsigned short integer?
a) -2147483648 to -2147483647 b) 0 to 4294967295 c) -32768 to 32767 d) 0 to 65535 ✓
- In C++ the expression `sum=sum+num` can also be written as:
a) `sum+=num` b) `sum=num++` ✓ c) `sum+=num` d) `num+=sum`
- Which of the following is equal to operator?
a) `+=` b) `==` ✓ c) `=` d) `++`
- Which of the following is an arithmetic operator?
a) `&&` b) `%` c) `<=` d) `++` ✓
- Which of the following operator is used to form compound condition?
a) Arithmetic operator b) Assignment operator c) Relational operator ✓ d) Logical operator
- The number of bytes reserved for a variable of data type float is:
a) 2 b) 4 c) 6 d) 8 ✓
- How cursor is moved to the next tabular position for displaying data?
a) By using reserved word b) By using reserved word
c) By using escape sequence ✓ d) By using header file
- A statement that starts with a `#` sign is called:
a) Preprocessor directive ✓ b) Keyword c) Comment d) Data type
- The name of header file is written between:
a) `[]` b) `()` c) `<>` ✓ d) `<<>>`
- Another name for keyword is:
a) Reserved words ✓ b) Special words c) Comments d) Identifier

12. Which is not a reserved word?
 a) int b) float c) break ✓ d) print ✓
13. C++ statement ends with:
 a) Period (.) b) Comma (,) c) Semicolon (;) ✓ d) Single quote (')
14. Variables are created in:
 a) RAM ✓ b) ROM c) Hard disk d) USB
15. What is the output produced by the statement? `Cout<< "\\nnow\\n";`
 a) "\\nnow" b) \\nnow c) "\\nnow" ✓ d) "\\nnow" ✓
16. Which of the following is a valid c++ statement?
 a) `char ch= 'a';` b) `char ch= 'ab';` c) `char ch=97` d) `char ch= "ab";`
17. Which of the following is a valid example of character constant?
 a) 'A' b) '9' ✓ c) '\$' d) All ✗
18. In c++, string constants are written in:
 a) Double quotes (" ") ✓ b) Single quotes (') c) Exclamation mark (!) d) Pound sign(\$)
19. A method to convert the data type of a value is called.
 a) Variable changing b) Type casting ✓ c) Value casting d) Type changing
20. The expression `3%5` has a value equal to:
 a) 3 ✓ b) 5 c) 0 d) None

Q2. Write short answers of the following questions.

1. Define reserved words and give three examples.

Ans. Reserved words are special words which are reserved by a programming language for specific purpose in program. These words cannot be used as variable names. All the reserved words are written in lower-case letters. Some examples of reserved words are *void*, *break*, *for*, *char*, and *case* etc. there are about 80 reserved words in C++ but it may vary depending on the version being used. Reserved words are also known as key words.

2. What is the purpose of using header file in program?

Ans. Header files are used in a program because the header files contain information that is required by the program in which these files are used. C++ contains many header files. These files have .h extension. Some example of headers files are *iostream.h*, *conio.h* and *math.h*. Header file must be included in the program in order to use its function in the program.

3. State the following variable names are valid or invalid. State reason for invalid variable names: a) `a123` b) `_abcd` c) `5html` d) `tot.al` e) `f3ss1` f) `c$avg` g) `net_weight` h) `cout`

Ans.

Sno	Variable Name	Valid / Invalid	Reason if invalid
1	<code>a123</code>	valid	
2	<code>_abcd</code>	valid	
3	<code>5html</code>	invalid	A variable name cannot start with a digit.
4	<code>tot.al</code>	invalid	A variable cannot have a period in it.
6	<code>fss1</code>	valid	
7	<code>c\$avg</code>	invalid	The special symbol \$ cannot be used in a variable name.
8	<code>net_weight</code>	valid	
9	<code>cout</code>	valid	

4. Why escape sequence is used? Give three examples with explanation.

Ans. Escape sequences are used to control the format of output. These characters are used in output statement. Three examples of escape sequences are as follows:

Escape sequence	Explanation	Programming code example
\n	Moves cursor to the beginning of next line	cout<< "Pakistan\nZindabad"; output: Pakistan Zindabad
\t	Moves cursor to the next horizontal tabular position	cout<<"Pakistan\tZindabad"; output: Pakistan Zindabad
\"	Produces double quote	cout<<"\"Pakistan Zindabad\""; output: "Pakistan Zindabad"

5. Differentiate between relational and logical operator.

Ans. The relational operators are used to compare two values of same types. After evaluate of a relational expression the result produced is True or False. ==, !=, >, >= are some of the examples of relational operator. The logical operators are used to combine multiple conditions or reverse a condition. AND, OR and NOT are logical operators used in C++.

6. What will be the output of the following statements?

- a. cout<< "17/2 is equal to"<<17/2; Ans: 8
b. cout<< "10.2/4 is equal to "<<10.2/4; Ans: 2.55
c. cout<< "40/5%3*7 is equal to"<<40/5%3*7; Ans: 14

7. Evaluate the following integer expressions.

- a) 3+4*5 Ans: 23 b) 4*5/10+8 Ans:10 c) 3*(2+7*4) Ans:90
d) 20-2/6+3 Ans: 23 e) (20-2)/(6+3) Ans: 2 f) 25%7 Ans: 4

8. Evaluate the following expressions that have integer and floating point data type.

- a) 10+15/2+4.3 Ans:21.3 b) 10.0+15.0/2+4.3 Ans:21.8 c) 4/6*3.0+6 Ans:6

9. What will be the output of the following program?

```
#include<iostream>
```

```
#include<conio.h>
```

```
void main()
```

```
{
```

```
int num1,num2,total;
```

```
num1=13;
```

```
num2=20;
```

```
total=num1+num2;
```

```
cout<< "The total of" <<num1<< "and"<<num2<< "s"<<total;
```

```
getch();
```

```
}
```

Output:

The total of 13 and 20 is 33

10. What will be the output of the following program?

```
#include<iostream>
```

```
#include<conio.h>
```

```

void main()
{
    int n;
    n=10;
    cout<< "The initial value of n is"<<n<<endl;
    n++;
    cout<< "The value of n is now"<<n<<endl;
    n++; n++;
    cout<< "The value of n is now"<<n<<endl;
    n--;
    cout<< "The value of n is now"<<n<<endl;
    getch();
}

```

Output:

```

The initial value of n is 10
The value of n is now 11
The value of n is now 13
The value of n is now 12

```

11. Determine the output of the following codes:

1) cout<< "Hello\n World \n Pakistan";	2) cout<< "\top ba\na\na";
3) cout<< "*\n**\n***\n";	4) int one=1,two=2; cout<<one<<two<< "Three";
5) int num=10; num+=5; num-=2; num*=3; cout<<++num;	6) int x=10; cout<<x<<endl; cout<<x--<<endl; cout<<--x;
7) int n; float x=3.8; n=int(x); cout<< "n="<<n<<endl;	8) int i=3; cout<<i; cout<<i++; cout<<++i;
9) int n1,n2,n3; n1=10; n2=n1++; n3=++n2; cout<<n1--<<endl; cout<<--n3<<endl; cout<<n1++;	10) int a,b; a=10; b=20; a=b; b=a; cout<<"a="<<a<<endl; cout<<"b="<<b<<endl;

Ans.

1) Ans. Hello World Pakistan	2) Ans. op ba a\na
3) Ans. * ** ***	4) Ans. 12three

5) Ans. 40	6) Ans. 10 10 8
7) Ans. n = 3	8) Ans. 224
9) Ans. 11 10 10	10) Ans. b=20 a=20

Q3. Write long answers of the following questions.

1. Define variable and write the rules for specifying variable names.

Ans. See Ans on Page 35

2. Why is type casting is used? Explain the types of type casting with an example of each.

Ans. See Ans on Page 46

3. Define endl and setw manipulators and give an example of each.

Ans. See Ans on Page 39

4. What is meant by precedence of operators? Write the operators with the highest precedence at the top and the lowest at the bottom.

Ans.

Precedence	Operator	Description
1.	*, /, %	Multiplication, Division and Remainder
2.	+, -	Addition and Subtraction
3.	<, <=, >, >=	Relational Operators
4.	=, !=	Equal to and Not Equal to
5.	!	Logical NOT
6.	&&	Logical AND
7.		Logical OR
8.	=, *=, /=, %=, +=, -=	Assignment Operators

Lab Activities / Programs

1. Write a program that reads four integers and print their sum, product and average.

Ans. `#include<iostream> // use iostream.h if you are using turbo or borland C++ IDE.
using namespace std; // do NOT this statement if you are using turbo or borland C++ IDE.
int main(void)
{

 int n1,n2,n3,n4,sum,prod;

 float avg;
 cout<<"Enter four integer numbers";
 cin>>n1>>n2>>n3>>n4;
 sum=n1+n2+n3+n4;
 prod=n1*n2*n3*n4;
 avg=sum/4;
 cout<<"Sum ="<<sum<<endl;
 cout<<"Product ="<<prod<<endl;
 cout<<"Average ="<<avg;
 return 0;
}`

2. Write a program that reads length and breadth of a rectangle and print its area.

Ans. `#include<iostream> // use iostream.h if you are using turbo or borland C++ IDE.
using namespace std; // do NOT this statement if you are using turbo or borland C++ IDE.
int main(void)
{
 int l,b,area;
 cout<<"Enter Length of a rectangle ";
 cin>>l;
 cout<<"Enter breadth of a rectangle ";
 cin>>b;
 area=l*b;
 cout<<"Area of a rectangle is "<<area<<endl;
 return 0;
}`

3. Write a program that reads temperature in Fahrenheit and prints its equivalent temperature in Celsius using the formula: $c = 5/9(f-32)$.

Ans. `#include<iostream> // use iostream.h if you are using turbo or borland C++ IDE.
using namespace std; // do NOT this statement if you are using turbo or borland C++ IDE.
int main(void)
{
 float c,f;
 cout<<"Enter temperature in Fahrenheit ";
 cin>>f;
 c=5.0/9*(f-32);
 cout<<"The temperature in Celsius is "<<c;
 return 0;
}`

UNIT 4

CONTROL STRUCTURES

CONTROL STRUCTURE

The structures/statements through which we can control the logical flow of a program is known as control structure. In a structured program, the logical flow is implemented three basic control structures as follows:

i) **Sequential Structure:** It refers to the execution of statements in order in which they appear. The statements are executed one after the other. No statement is skipped and no statement is executed more than once. These types of programs are also known as "straight line" programs.

ii) **Conditional Structure:** The conditional structure executes a statement or set of statements based on a condition. It executes a statement if the given condition is true otherwise it ignores the statement if the condition is false. It is also called decision making/control structure or selection structure. Some of the selection structures are If, else-if, and switch etc.

iii) **Iteration Structure:** The iteration structure provides the way to repeat one or more statements a specific number of times. It is also known as repetition. Loops are the example of iteration structure.

DECISION CONTROL STRUCTURE

Decision control structures are used in programming language to make decisions. They allow programs to execute a specific statement or a set of statements based on one or more conditions. Following are the decision making structures used in C++ language.

1- if 2- if-else 3- else-if 4- switch

1- IF

The basic if statement allows a program to execute a single or block of statements if a given condition is true.

Syntax / General Form of If

```
if(condition)
```

```
{
    block of statements
}
```

(The braces are not required if there is a single statement to be executed)

Disadvantage:

The 'if' statement is very limited in its use. The statement or set of statements is executed if the condition is true, but if the condition is false then nothing happens.

Example program that reads marks from user and print the message "You have passed" if marks are above 32.

```
#include <iostream>
using namespace std;
int main(void)
{
    int marks;
    cout<< "Enter your marks";
    cin>>marks;
    if(marks>32)
        cout<< "You have passed";
    return 0;
}
```

Output:

```
Enter your marks 38
You have passed
```

2- IF-ELSE

The 'if else' statement allows one block of statements to be executed when the condition is true and another to be executed when the condition is false.

Syntax / General Form of If...else

```
if(condition)
{
    block of statements
}
else
{
    block of statements
}
```

Example program that reads marks from user and print "Congratulations" "You have passed" if marks are above 32 otherwise it prints "Better luck next time" "You have Failed".

```
#include<iostream>
using namespace std;
int main(void)
{
    int marks;
    cout<< "Enter your marks";
    cin>>marks;
    if(marks>32)
    {
        cout<< "Congratulations";
        cout<< "You have passed";
    }
    else
    {
        cout<< "Better luck next time";
        cout<< "You have failed";
    }
}
```

Output:

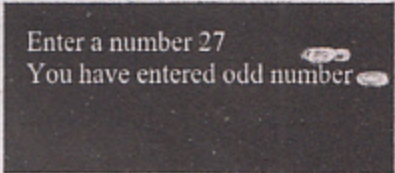
```
Enter your marks 21
Better luck next time
You have failed
```

```
}  
return 0;
```

Example program that reads a number and prints whether it is even or odd number.

```
#include<iostream>  
using namespace std;  
int main(void)  
{  
    int num,remainder;  
    cout<< "Enter a number";  
    cin>>num;  
    remainder=num%2;  
    if(remainder==0)  
        cout<< "You have entered even number";  
    else  
        cout<< "You have entered odd number";  
}  
return 0;  
}
```

Output:



```
Enter a number 27  
You have entered odd number
```

Nested If

If statement within if statement is called nested if statement. In nested if structure, the control enters into the inner if only when the outer if condition is true. Only one block of statements are executed and the remaining blocks are skipped.

Syntax / General Form of Nested If

```
if(condition)  
{  
    if(condition)  
    {  
        statements;  
    }  
    else  
    {  
        statements;  
    }  
}  
else  
{  
    statements;  
}
```

} **nested if**

Example program that inputs three numbers and displays the largest using nested if.

```
#include<iostream>
using namespace std;
void main(void)
{
    int n1,n2,n3;
    cout<<"Enter three Numbers: ";
    cin>>n1>>n2>>n3;
    if(n1>n2)
    {
        if(n1>n3)
            cout<<n1<<" is largest number";
        else
            cout<<n3<<" is largest number";
    }
    else
    {
        if(n2>n3)
            cout<<n2<<" is largest number";
        else
            cout<<n3<<" is largest number";
        }
    return 0; }
```

Output:

```
Enter the numbers18 6 19
19 is largest number
```

3- ELSE-IF

In this structure the program checks the condition from the beginning. If any of the condition is true, the program executes the statements under the 'if' or 'else if'. If none of the condition is true, then the program executes the statements following the final else. It has the following general form.

Syntax / General Form of else-if

```
if(condition-1)
{
    block of statements
}
else if(condition-2)
{
    block of statements
}
else if(condition-3)
{
    block of statements
}
else
{
    block of statements to be executed if none of
    the above condition is true
}
```

Example program to perform simple arithmetic operations by using "else-if"

```
#include<iostream>
using namespace std;
int main(void)
{
    int n1,n2; char opt;
    cout<<"Enter 1st value: ";
    cin>>n1;
    cout<<"Enter 2nd value: ";
    cin>>n2;
    cout<<"Enter operator: ";
    cin>>opt;
    cin>>n1>>opt>>n2;
    if(opt== '+')
    {
        cout<<"\nAddition= "<<n1+ n2;
    }
    else if(opt== '-')
    {
        cout<<"\nSubtraction= "<<n1- n2;
    }
    else if(opt== '*')
    {
        cout<<"\nMultiplication= "<<n1* n2;
    }
    else if(opt== '/')
    {
        cout<<"\nDivision= ",n1/ n2;
    }
    else if(opt== '%' )
    {
        cout<<"\nRemainder ",n1% n2;
    }
    else
    {
        cout<<"\n you entered invalid operator";
    }
    return 0;
}
```

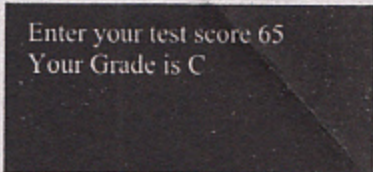
Output:

```
Enter 1st value: 2
Ent 2nd value: 6
Enter operator: *
Multiplication=20
```

Program that inputs test score of a student and displays his grade according to the given criteria.

Score	Grade
80-100	A
70-79	B
60-69	C
50-59	D
below 50	F

```
#include<iostream>
using namespace std;
int main(void)
{
    int s;
    cout<<" Enter your test score";
    cin<<s;
    if(s>=80 && s<=100)
        cout<<"\n Your Grade is A";
    else if(s>=70 && s<=79)
        cout<<"\n Your Grade is B";
    else if(s>=60 && s<=69)
        cout<<"\n Your Grade is C";
    else if(s>=50 && s<=59)
        cout<<"\n Your Grade is D";
    else
        cout<<"\n Your Grade is F";
    return 0;
}
```

Output:

```
Enter your test score 65
Your Grade is C
```

4- SWITCH

The Switch statement is similar to else if structure. It enables the program to select from multiple choices on a set of fixed values. The choices are called cases. We specify the possible **choices** in a switch statement using as many cases as we need. The last case of switch statement is default case and the break statement in default case is optional.

Syntax / General Form of switch

```
switch(variable/expression)
```

```
{
```

```
case constant-1: statements
```

```
    break;
```

```
case constant-2: statements
```

```
    break;
```

```
case constant-3: statements
```

```
    break;
```

```
case constant-n: statements
```

```
    break;
```

```
default: statements to be executed when none of the above case is true.
```

```
}
```

The **break** statement is necessary to terminate the switch statement when the statements of a particular case have been executed. The break statement is optional in the default statement.

Program that inputs an alphabet and checks whether it is vowel or consonant.

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
    char ch;
    cout<<"Enter an alphabet ";
    ch=getche();
    switch(ch)
    {
        case 'a':
        case 'A': cout<<ch<<"is a vowel"; break;
        case 'e':
        case 'E': cout<<ch<<"is a vowel"; break;
        case 'i':
        case 'I': cout<<ch<<"is a vowel"; break;
        case 'o':
        case 'O': cout<<ch<<"is a vowel"; break;
        case 'u':
        case 'U': cout<<ch<<"is a vowel"; break;
        default: cout<<ch<<"is a consonant";
    }
    return 0;
}
```

Output:

```
Enter an alphabet B
B is a consonant
```

Program that reads an integer between 1 to 7 that represent a day of week starting from Monday. It prints the name of the day based on the value of day.

```
#include<iostream>
using namespace std;
int main()
{
    int day;
    cout<<"Enter an integer (1-7): ";
    cin>>day;
    switch(day)
    {
        case 1: cout<<"\nMonday"; break;
        case 2: cout<<"\nTuesday"; break;
        case 3: cout<<"\nWednesday"; break;
        case 4: cout<<"\nThursday"; break;
        case 5: cout<<"\nFriday"; break;
        case 6: cout<<"\nSaturday"; break;
        case 7: cout<<"\nSunday"; break;
        default: cout<<"\nNot a valid day";
    }
    return 0;
}
```

Output:

```
Enter an integer (1-7): 4
Thursday
```

DIFFERENCE BETWEEN ELSE-IF AND SWITCH STATEMENT

ELSE-IF	SWITCH
1- It uses multiple expressions for multiple choices.	It uses single expression for multiple choices.
2- It can check a range of values.	It cannot check a range of values.
3- It evaluates integer, character, or logical expression.	It evaluates only character of integer values.
4- It can test for equality as well as for logical expression.	It can only test for the equality.

LOOPS

Loop is a control structure that is used to repeat a statement or set of statements a number of times until the condition is true. Loops are also known as repetition control structures. Following are the types of loop used in C++.

1- For Loop 2- While Loop 3- Do While Loop

1- FOR LOOP

For loop is a type of loop that is used when the number of repetitions are known or predetermined before the execution of the program. It is also called counter loop or counter-controlled loop.

Syntax / General form of For Loop

```
for(expr1 / initialization ; expr2 / condition ; expr3 / increment or decrement)
{
    Body of loop
}
```

Initialization: It specifies the starting value of counter variable. One or more than one variable can be initialized in this part.

Condition: The condition is given as a relational expression. The body statements are executed as far as the given condition is true. If the condition is false the statements are not executed.

Increment/decrement: This part of loop specifies the change in counter variable, after each execution of the loop.

Statement: Statement is the instruction that is executed when the condition is true. If more than one statement is to be repeated, then these statements must be enclosed with braces i.e. { }. These statement / statements are also known as body of loop.

Example program of For Loop.

```
#include<iostream>
using namespace std;
int main()
{
    int i;
    for(i=1;i<=5;i++)
```

Output:

```
Hello Word
Hello Word
Hello Word
Hello Word
Hello Word
```

```
cout<<"\nHello World";  
return 0;  
}
```

Explanation / Working of for loop

In the above example, the variable *i* is initialized with 1. The termination condition is $i \leq 5$. It means that the loop will continue while the value of variable *i* is less than or equal to 5. The loop will terminate when the value of variable *i* becomes greater than 5. The third part i.e. increment indicates that the value of *i* will be incremented by 1 after each execution of loop.

Program to print 1,2,3,4,5,6,7,8,9,10 series.

```
#include<iostream>  
using namespace std;  
int main()  
{  
    int i;  
    for(i=1;i<=10;i++)  
        cout<<i<<" ";  
    return 0;  
}
```

Output:

1,2,3,4,5,6,7,8,9,10,

Program to print all the positive odd numbers that are less than twenty (1 3 5 7 9 11 13 15 17 19) on a single line.

```
#include<iostream>  
using namespace std;  
int main()  
{  
    int i;  
    for(i=1;i<20;i+=2)  
        cout<<i<<" ";  
    return 0;  
}
```

Output:

1 3 5 7 9 11 13 15 17 19

Program to print 100,90,80,70,60,50 series.

```
#include<iostream>  
using namespace std;  
int main()  
{  
    int i;  
    for(i=100;i>=50;i-=10)  
        cout<<i<<" ";  
    return 0;  
}
```

Output:

100,90,80,70,60,50,

Program that inputs a number and print its multiplicative table up to 20

```
#include<iostream>
using namespace std;
int main(void)
{
    int n,i;
    cout<<"Enter number to print its table";
    cin>>n;
    for(i=1;i<=20;i++)
    cout<<endl<<n<<"x"<<i<<"="<<n*i;
    return 0;
}
```

Output:

```
Enter number to print its table 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
...
5 x 20 = 100
```

Program that inputs a number from the user and displays the factorial of that number using for loop.

```
#include<conio.h>
#include<iostream>
using namespace std;
int main(void)
{
    int n,i;
    int f=1;
    cout<<"Enter number to print its factorial";
    cin>>n;
    for(i=1;i<=n;i++)
    f=f*i;
    cout<<"\nFactorial of "<<n<<" is"<<f;
    return 0;
}
```

Output:

```
Enter number to print its factorial 4
Factorial of 4 is 24
```

Program to print a table of equivalent temperature from 50 to 100 in Fahrenheit and Celsius with an increment of 5. (Formula of conversion is $c=0.55*(f-32)$).

Output:

```
#include<iostream>
using namespace std;
int main(void)
{
    int fah;
    float cel;
    cout<<"\n Fahrenheit    Celsius\n";
    for(fah=50;fah<=100;fah+=5)
    {
        cel= 0.55 * (fah-32);
        cout<<"\n"<<fah<<"\t"<<cel;
    }
    return 0;
}
```

Fahrenheit	Celsius
50	9.90
55	12.65
60	15.40
65	18.15
70	20.90
...	...
100	37.40

*\\Here two statements are to be executed that's why
\\ the body of for loop is made compound by putting { }*

2- WHILE LOOP

While loop is useful when the number of iterations/repetitions are unknown before the execution of program or when the terminating condition occurs unexpectedly. It is also known as conditional loop or sentinel loop.

```
|  
Syntax / General form of While Loop  
while(condition/check)  
{  
    Body of loop  
}
```

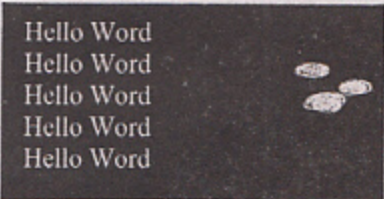
Condition: The condition is given as a relational expression. The statement is executed as far as the given condition is true. If the condition is false the statement is not executed.

Statement: Statement is the instruction that is executed when the condition is true. If more than one statement is to be repeated, then these statements must be enclosed with braces i.e. { }. These statement / statements are also known as body of loop.

Example program of While Loop.

```
#include<iostream>  
using namespace std;  
int main()  
{  
    int i;  
    i=1;  
    while(i<=5)  
    {  
        cout<<"\nHello World";  
        i++;  
    }  
    return 0;  
}
```

Output:



```
Hello World  
Hello World  
Hello World  
Hello World  
Hello World
```

Explanation / Working of while loop

In the above example, the variable *i* is initialized with 1 (the variable is always initialized outside of the body of the loop). The termination condition is *i* <= 5. It means that the loop will continue while the value of variable *i* is less than or equal to 5. The loop will terminate when the value of variable *i* becomes greater than 5. The body of loop contains two statements. The first statement is *cout* and the second statement is increment that indicates that the value of *i* will be incremented by 1. After increment the control moves back to the condition. This process continues for five times.

Program that converts kilogram to pounds using while loop. In this program zero signals the termination of loop.

```
#include<iostream>
using namespace std;
int main()
{
    float kgs,lbs;
    cout<<"Enter weight in kilogram (0 to end):";
    cin>>kgs;
    while(kgs!=0)
    {
        lbs=kgs*2.2;
        cout<<endl<<kgs<<"kilograms ="<<lbs<<"pounds";
        cout<<"Enter weight in kilogram (zero to end)";
        cin>>kgs;
    }
    cout<<"\n Thank you";
    return 0;
}
```

Output:

```
Enter weight in kilogram (0 to end):62
62 kilogram = 136.4000000 pounds
```

```
Enter weight in kilogram (0 to end):50
52 kilogram = 118.8000000 pounds
```

```
Enter weight in kilogram (0 to end): 0
Thank you
```

3- DO WHILE LOOP

Do while loop is similar to while loop, the only difference between the do while loop and other loops is that, in do while loop the body of the loop executes at least once whether the condition is true or false because in this loop the condition appears after the body of loop.

Syntax / General form of Do While Loop

```
do
{
    body of loop
} while(condition/check);
```

do: It is a keyword that indicates the beginning of the loop.

Condition: The condition is given as a relational expression. The statement is executed as far as the given condition is true. If the condition is false the statement is not executed.

Statement: Statement is the instruction that is executed when the condition is true. If more than one statement is to be repeated, then these statements must be enclosed with braces i.e. { }. These statement / statements are also known as body of loop.

In do-while loop the condition will always ends with a semicolon. An error occurs if the semicolon is not used at the end of the condition.

Example program of Do While Loop.

```
#include<iostream>
using namespace std;
int main()
{
    int i;
    i=1;
    do
    {
```

Output:

```
Hello Word
Hello Word
Hello Word
Hello Word
Hello Word
```

```

cout<<"\nHello World";
i++;
}
while(i<=5);
return 0;
}

```

Explanation / Working of do while loop

First of all, the body of loop is executed. After executing the statements in loop body, the condition is checked. If it is true, then the control again enters the body of the loop and executes the statements in the body again. This process continues as long as the condition remains true. The loop terminates when the condition becomes false. This loop is executed at least once, even if the condition is false in the beginning.

Program that inputs two numbers and displays addition of two numbers then ask user to press 'Y' to continue or press any other key to quit.

```

#include<iostream>
using namespace std;
int main()
{
    int num1,num2;
    char option='y';
    do
    {
        cout<<"Enter First Number:";
        cin>>num1;
        cout<<"Enter Second Number:";
        cin>>num2;
        cin<<"\n"<<num1<<"+ "<<num2<<"="<<num1+num2;
        cout<<"\nPress y to continue or any other key to quit:";
        option=getche();
    }
    while(option=='y');
    cout<<"\nGood Bye";
    return 0;
}

```

Output:

```

Enter First Number: 10
Enter Second Number: 20
10 + 20 = 30
Press y to continue or any other key to quit: y
Enter First Number: 80
Enter Second Number: 30
80 + 30 = 120
Press y to continue or any other key to quit: n
Good Bye

```

DIFFERENCE BETWEEN WHILE LOOP AND DO-WHILE LOOP

WHILE LOOP	DO-WHILE LOOP
1- It is a pre-tested loop, because the condition is checked before the body of the loop.	It is a post-tested loop, because the condition is checked after the body of the loop.
2- The body of the loop does not executes if the condition is false in the beginning.	The body of the loop executes at least at once even if the condition is false in the beginning.
3- It is called as entry controlled loop.	It is called as exit controlled loop.
4- The semicolon is not used after the condition.	The semicolon is used after the condition.

THE BREAK AND CONTINUE STATEMENT

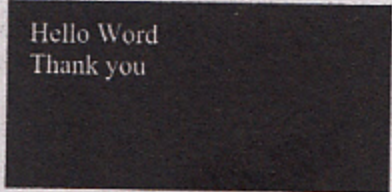
1- The break statement: The break statement can be used in two ways in the program as follows:

- It can be used in switch statement to terminate a case. The program continues to execute the next statement after switch statement.
- It can be used to terminate a loop when it is used inside the loop body of loop. When break statement is executed in the loop body, the remaining iterations of the loop are skipped. The program continues to execute next statement after the loop.

Example program of break statement.

```
#include<iostream>
using namespace std;
int main()
{
    int a;
    for(a=1;a<=10;a++)
    {
        cout<<"\nHello World";
        break;
        cout<<"\n C++ is very interesting ";
    }
    cout<<"Thankyou";
    return 0;
}
```

Output:



```
Hello World
Thank you
```

Explanation

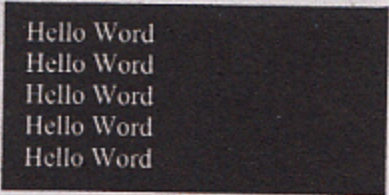
The above program uses break statement in "for loop". The counter variable 'a' indicates that the loop should execute five times. But it is executed only once. In the first iteration, the cout statement in the loop displays "Hello World" and then the control moves to "break" statement. This statement moves the control out of the loop and the remaining iterations are skipped and the message appears only once.

2- The continue statement: The continue statement is used inside the body of the loop. It is used to transfer the control to the beginning (increment/decrement) of the loop. When continue statement is executed inside the loop body, the remaining statements of the current iteration are not executed. The control directly move to the next iteration.

Example program using "continue" statement.

```
#include<iostream>
using namespace std;
int main()
{
    int a;
    for(a=1;a<=5;a++)
    {
        cout<<"\nHello World";
        continue;
        cout<<"\n C++ is very interesting to learn";
    }
}
```

Output:



```
Hello World
Hello World
Hello World
Hello World
Hello World
```

```

    }
    return 0;
}

```

Explanation

The above program uses two cout statements inside the "for loop". One cout statement is before continue statement and the other is after continues statement. The second cout statement is never executed because each time continue statement is executed, the control transfer back to the beginning of the loop. So "The C++ is very interesting to learn" is never displayed in this example.

The exit() function

The exit() function is used to terminate a C++ program before its normal termination. It transfer the control back to operating system. stdlib.h header file is included in the program in order to use exit() function. The syntax of exit() function is as follows:

```
exit(value);
```

here value indicates the exit() code. It is given as integer constant or variable. The exit code 0 exits the program without any error. The exit code 1 indicates that the program was terminated due to an error. It helps the programmer in debugging the program.

Example:

```

cout<< " Enter an integer";
cin>>n;
if(n>0)
cout<< "You entered positive integer";
else
exit(0);

```

NESTED LOOP

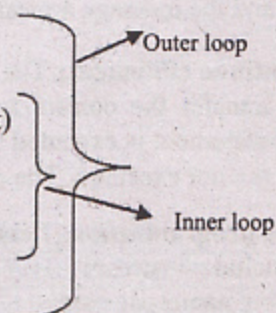
A loop within another loop is known as nested loop or internal loop. Nested loop can be of any kind. We can nest for loop inside a while loop or a do while loop or vice versa.

Syntax / General form of Nested For loop

```

for(initialization;condition;increment/decrement)
{
    for(initialization;condition;increment/decrement)
    {
        Body of loop
    }
}

```



Here it should be noted that the outer loop is executed first and Inner/Nested loop is completed first.

Example program of Nested for Loop.

```
#include<iostream>
using namespace std;
int main()
{
    int i, j;
    for(i=1; i<=3; i++)
    {
        for(j=1; j<=3; j++)
        {
            cout<<"\n"<<i<<" "<<j;
        }
    }
    return 0;
}
```

Output:

```
1,1
1,2
1,3
2,1
2,2
2,3
3,1
3,2
3,3
```

Example program of Nested for Loop.

```
#include<iostream>
using namespace std;
int main()
{
    int i, j;
    for(i=1; i<=5; i++)
    {
        for(j=1; j<=5; j++)
        {
            cout<<j<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

Output:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Example program of Nested for Loop.

```
#include<iostream>
using namespace std;
int main()
{
    int i, j;
    for(i=1; i<=5; i++)
    {
        for(j=1; j<=i; j++)
        {
            cout<<j;
        }
        cout<<endl;
    }
    return 0;
}
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Example program of Nested for Loop.

```
#include<iostream>
using namespace std;
int main()
{
    int i, j;
    for(i=1; i<=5; i++)
    {
        for(j=i; j<=5; j++)
        {
            cout<<j;
        }
        cout<<endl;
    }
    return 0;
}
```

Output:

```
1 2 3 4 5
2 3 4 5
3 4 5
4 5
5
```

Example program of Nested for Loop.

```
#include<iostream>
using namespace std;
int main()
{
    int i, j;
    for(i=5; i>=1; i--)
    {
        for(j=1; j<=i; j++)
        {
            cout<<"\t"<<j;
        }
        cout<<"\n";
    }
    return 0;
}
```

Output:

```
5 5 5 5 5
4 4 4 4
3 3 3
2
1
```

Example program of Nested while Loop.

```
#include<iostream>
using namespace std;
int main()
{
    int i, j;
    i=1;
    while(i<=5)
    {
        j=1;
        while(j<=5)
        {
            cout<<"\t*";
            j++;
        }
        cout<<endl;
        i++;
    }
}
```

Output:

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

```

    }
    cout<<"\n";
    i++;
}
return 0;
}

```

Example program of Nested while Loop.

```

#include<iostream>
using namespace std;
int main()
{
    int i, j;
    i=1;
    while(i<=5)
    {
        j=1;
        while(j<=i)
        {
            cout<<"\t*";
            j++;
        }
        cout<<"\n";
        i++;
    }
    return 0;
}

```

Output:

```

*
* *
* * *
* * * *
* * * * *

```

EXERCISE

Q1. Select the best answer for the following MCQs.

- How is a single statement for loop terminated?
 - a) with a colon
 - b) with a right brace
 - c) with a right bracket
 - d) with a semicolon
- How is multiple statement for loop terminated?
 - a) with a colon
 - b) with a right brace
 - c) with a right bracket
 - d) with a semicolon
- Which of the following can be used to replace ternary operator?
 - a) if-else statement
 - b) break statement
 - c) else-if statement
 - d) while loop
- Which of the following can be used to replace switch statement?
 - a) if-else statement
 - b) break statement
 - c) else-if statement
 - d) while loop
- A while loop is more appropriate to use than a for loop when:
 - a) the body of the loop is executed at least once
 - b) the termination condition occurs unexpectedly
 - c) the program executes at least once
 - d) the number of iterations are known in advance
- In which situation a do while loop is more appropriate to use?
 - a) when the body of the loop is executed at least once
 - b) when the loop terminates unexpectedly
 - c) when the program executes at least once
 - d) when the number of iterations are known in advance
- In which situation a for loop is more appropriate to use?
 - a) when the body of the loop is executed at least once
 - b) when the loop terminates unexpectedly
 - c) when the program executes at least once
 - d) when the number of iterations are known in advance

8. Which of the following transfers control to the beginning of the loop, skipping the remaining statements?
a) exit function b) continue statement c) break statement d) nested loop
9. It is not part of switch statement:
a) default b) break c) case d) else
10. Which of the following are valid case statements in a switch?
a) case 1: b) case x<4 c) case 'ab' d) case 1.5
11. These are used to repeat the execution of specific block of statements:
a) Decision statements b) Loop structure c) Switch d) if...else
12. In which loop the condition comes after the body of the loop?
a) while b) do while c) for d) b and c
13. A loop with in a loop is called:
a) Nested loop b) Complex loop c) Infinite loop d) Dual loop
14. A for loop contains three expressions: initialization, conditions and:
a) Assignment b) Validation c) Increment/decrement d) Recalling
15. In a for loop, this expression is executed only once:
a) Condition b) Validation c) Initialization d) All
16. This means to increase a value of variable by one:
a) Modulus b) Increment c) Decrement d) Up
17. If you want a user to enter exactly 20 values, which loop would be best to use?
a) while b) do while c) for d) infinite
18. Semicolon is placed at the end of condition in:
a) while b) do while c) for d) All
19. Which of the following is loop statement?
a) if b) if-else c) Switch d) for
20. If the value of a=15 and b=10 then what will be the output of the expression a>b? a-b :a+b;
a) 15 b) 25 c) 5 d) 150

Q2. Write short answers of the following questions.

1. Why control statements are used in C++ programs?

Ans. The control statements are used in C++ programs to control the flow of execution in a program. Three types of control structures are used in programming i.e sequential structure, conditional structure, and iteration structure. The **sequential structure** executes statements one by one in the sequence in which they appear in the program from top to bottom till last instruction. The **conditional structure** which is also known as decision making structure is used to execute the statements based on condition. If the condition is true then a specific set of instructions are executed otherwise control is transferred to some other part of the program. The **iteration structure** also known as loop is used to repeat a statement or set of statements a number of times.

2. Differentiate between if-else and else-if statements.

Ans. The **if-else** statement allows making decision between two courses of action based on condition. One block of statements is executed if the condition is true and the other block is executed if the condition is false.

If-else-if statement can be used to choose one block of statements from many blocks of statements. It is used in situation where a decision is to be made from several alternatives based on various conditions.

3. Differentiate between else-if and switch.

Ans.

Else-if	Switch
1) Else-if statement test for equality as well as for logical expression.	Switch statement test only for equality.
Else-if statement evaluates integer, character, or logical expression.	Switch statement evaluates only character or integer value.
3) Either if statement will be executed or else statement is executed.	Switch statement executes one case after another till a break statement is appeared or the end of switch statement is reached.
4) If the condition inside if statement is false, then by default the else statement is executed.	If the condition inside switch statements does not match with any of cases, for that instance the default statement is executed.

4. Differentiate between for and while loop.

Ans. For loop is used when the number of iterations are known in advance before the execution of program. For loop is also known as counter loop. General form of for loop is as follows:

```
for(initialization;condition;increment/decrement)
```

```
{
    Block of statements
}
```

The while loop is used when the number of iterations are not known in advance before the execution of program. While loop is also known as conditional or sentinel loop. General form of for loop is as follows:

```
while(condition)
```

```
{
    Block of statements
}
```

5. Differentiate between while loop and do while loop.

Ans. See Ans on page 66

6. What is the usage of break and continue statements in C++ programs.

Ans. See Ans on page 67

7. What is the purpose of exit function?

Ans. See Ans on page 68

8. What is nested loop? Give one example.

Ans. See Ans on page 68

9. Write the following code using while loop.

```
sum=0;
for(k=20;k<100;k=k+2)
    sum=sum+k;
cout<< "\n"<<sum;
```

Ans. sum=0; k=0;
while(k<100)
{
 sum=sum+k;

```

    k=k+2;
}
cout<< "\n"<<sum;

```

10. Write the following code using switch statement, to produce the same output.

```

if(choice==1)
    cout<< "\n Sum="<<x+y;
else if(choice==2)
    cout<< "\n Product="<<x*y;
else
    cout<< "\n Average="<<(x+y)/2;

```

Ans.

```

switch(choice)
{
case 1: cout<<"Sum="<<x+y; break;
case 2: cout<<"Product="<<x*y; break;
default: cout<<"Average="<<(x+y)/2;
}

```

11. What will be the output of the following code?

```

int k=1,sum=0;
while(k<10)
{
    sum=sum+k;
    cout<<k<<"\t"<<sum<<endl;
    k=k+2;
}

```

Output:

```

1      1
3      4
5      9
7      16
9      25

```

12. What will be the output of the following code?

```

int a,b,c;
a=0;b=1;c=2;
a=b+c;
b=++a;
c=b++;
cout<<"a="<<a<<"\n"<<"b="<<b<<"\n"<<"c="<<c;

```

Output:

```

A=4
B=5
C=4

```

13. What will be the output of the following code?

a)

```

1 2 3 4 5
1 2 3 5
1 2 3
1 2
1

```

b)

```

1 2 3 4 5
2 3 4 5
3 4 5
4 5
5

```

c)

```

*
* *
* * *
* * * *
* * * * *

```

See Ans on Page 69

See Ans on Page 70

See Ans on Page 71

14. Determine the output of the following codes:

```

1) int a=1;
   int b=6;
   if(a+b<7)
       cout<<a;
   else

```

```

2) int p,q,r;
   if(x>=10)
   { if(x+y<40)

```

cout<<b;	{ y++; } else { y--; } cout<<x<<" "<<y;
3) int x=1, y=0; p=0; q=3; if(p%q==3) r=0; else r=1; cout<<r;	4) int x=5+7/2; switch(x) { case 5:cout<<"p"; break; case 7:cout<<"q"; break; case 8:cout<<"r"; break; default:cout<<"no output"; }
5) int m=5, c=0; while(c<100) { m=m+1; c=c+1; } cout<<m;	6) int n=45; int m=32; while(m>0) { cout<<n/m; n%=m; n/=2; }
7) char c='A'; do { cout<<c<<" "; c=c+2; } while(c<='I'); cout<<endl;	8) int i, p=1; for(i=1; i<6; i+=1) { p*=2; } cout<<"\np is="<<p;
9) for(int i=1; i<30; i+=3) { if(i%2==1) continue; else if(i==10) break; cout<<i<<endl; }	10) int i; for(i=1; i<=3; i++) cout<<"hi"; cout<<i;

Ans.

Ans. 6	2) Ans. 10 39
3) Ans. 1	4) Ans. r
5) Ans. 105	6) Ans. 101101

7) Ans. A C E G I	8) Ans. p is = 32
9) Ans. 4	10) Ans. hi hi hi 4

Q3. Write long answers of the following questions.

1. What is decision control structures? Explain all types of if statements with syntax and examples.

Ans. See Ans on Page 54,55

2. Explain the purpose of switch statement with syntax and one example.

Ans. See Ans on Page 59,60

3. What is looping control structure? Explain all types of looping statements with syntax and examples.

Ans. See Ans on Page 61

Lab Activities / Programs

1. Write a program that reads a number and prints its square if the number is greater than 10 otherwise prints its cube.

Ans.

```
#include<iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter a number ";
    cin>>num;
    if(num>10)
        cout<<"\nThe square of number is"<<num*num;
    else
        cout<<"\nThe cube of number is"<<num*num*num;
    return 0;
}
```

2. Write a program that reads an integer and prints whether it is odd or even number.

Ans. See Ans on Page 56

3. Write a program that reads three numbers and prints the largest one.

Ans. See Ans on Page 57

4. Write a program that reads a letter and prints whether it's lowercase or uppercase letter.

Ans

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
```

```

{
char ch;
cout<<"Enter a character letter: ";
ch=getche();
if(ch>='a' && ch<='z')
cout<<"\nyou entered Lower case letter";
else if(ch>='A' && ch<='Z')
cout<<"\nyour entered Upper case letter";
return 0;
}

```

5. Write a program that reads an integer and prints its multiplicative table up to 20.

Ans. See Ans on page 63

6. Sameer works in a firm as a programmer and is getting a month pay. Write a program to take his basic pay as input through the keyboard and calculate his house rent and Net pay. The house rent is to be calculate according to the scheme given below:

Basic pay	House Rent
Basic pay < 30000	30% of Basic pay
Basic pay >=30000 and <=50000	35% of Basic pay
Basic pay > 50000	40% of Basic pay

The Net pay is to be calculated as: $\text{net_pay} = \text{basic_pay} + \text{house_rent}$.

Ans.

```

#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
int basic_pay,net_pay,house_rent;
cout<<"Enter your basic pay ";
cin>>basic_pay;
if(basic_pay<3000)
house_rent=basic_pay*30/100;
else if(basic_pay>=30000 && basic_pay<=50000)
house_rent=basic_pay*35/100;
else
house_rent=basic_pay*40/100;
net_pay=basic_pay+house_rent;
cout<<"\nBasic Pay ="<<basic_pay;
cout<<"\nHouse Rent Pay ="<<house_rent;
cout<<"\nNet Pay ="<<net_pay;
return 0;
}

```

7. Write a program that will produce a table of equivalent temperatures in both Fahrenheit and Celsius with an increment of 5 from 50 to 100 as follows: $C = 5/9(F - 32)$

Fahrenheit	Celsius
50	9.90
55	12.65
.	.
100	37.40

Ans. See Ans on page 63

8. Write a program that print the sum of the following sequence using for loop.
Sum = 30 + 33 + 36 + 39 + ... + 60.

Ans.

```
#include<iostream>
using namespace std;
int main()
{
    int i,sum=0;
    for(i=30;i<=60;i+=3)
        sum=sum+i;
    cout<<"\n The sum is "<<sum;
    return 0;
}
```

9. Write the above program using while loop.

Ans.

```
#include<iostream>
using namespace std;
int main()
{
    int i=30,sum=0;
    while(i<=60)
    {
        sum=sum+i;
        i=i+3;
    }
    cout<<"\n The sum is "<<sum;
    return 0;
}
```

10. Write a program that prints all the positive odd numbers up to 50 skipping those that are divisible by 5 using continue statement.

Ans.

```
#include<iostream>
using namespace std;
int main()
{
    int i;
    for(i=1;i<=50;i+=2)
```

```

{
    if(i%5==0)
        continue;
    cout<<j<<"\t";
}
return 0;
}

```

11. Write a program that reads an integer and print its factorial.

Ans. See Ans on page 63

12. Write a program that read coefficients a,b and c of the quadratic equation $ax^2+bx+c=0$ and prints the real solution of x, using the following formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Note that if $b^2-4ac = 0$, then there is only one real solution. If it is greater than zero then there are two real solutions and if it is less than zero then print the message "NO REAL SOLUTION"

2.
Ans.

```

#include<iostream>
#include<math.h>
using namespace std;
int main()
{
    float a,b,c,x1,x2,r;
    cout<<"Enter three coefficients ";
    cin>>a>>b>>c;
    r=b*b-4*a*c;
    if(r==0)
    {
        x1=-b+sqrt(r)/(2*a);
        cout<<"\nThe only Real solution is"<<x1;
    }
    else if(r>0)
    {
        x1=-b+sqrt(r)/(2*a);
        x2=-b-sqrt(r)/(2*a);
        cout<<"\nThe 1st Real Solution is"<<x1;
        cout<<"\nThe 2nd Real solution is"<<x2;
    }
    else
        cout<<"\nNo Real Solutions";
    return 0;
}

```

UNIT 5

ARRAYS AND STRINGS

ARRAY

Collection of data items of same type is known as array. The data items are stored in contiguous memory locations. Array allows programmer to use a single variable name to represent a collection of same type of data. The individual element of an array is referred by using an integer that is known as an index or subscript. The index/subscript of an array always starts with 0.

Arrays are used to store a large amount of same kind of data. Suppose the user wants to store marks of 100 students. For that user declares 100 variables. It will be a very time consuming and complex process to use these 100 variables individually. This process can be simplified by using array. Any array of 100 elements can be used to store and process these 100 values very easily.

Memory representation of array 'Marks' storing 5 elements.

Marks		← Name of Array
22	0	← Index of array starting from 0
51	1	
60	2	
81	3	
90	4	

ADVANTAGES OF USING ARRAY

- 1- Arrays can store a large number of values with single name.
- 2- Arrays are used to process a list of numbers and strings easily and quickly.
- 3- Different operations such as sorting, searching etc can be performed on the values stored in an array.
- 4- The use of array reduces program size.

TYPES OF ARRAY

There are two types of array used in C++.

- 1- One-Dimensional Array
- 2- Multi (two) Dimensional Array

1- ONE DIMENSIONAL ARRAY

In one dimensional array all the elements are arranged in the form of list. It is also known as linear array or list. It consists of only one subscript or index value to refer an individual element of an array. It has only one column or one row.

DECLARATION / DEFINITION OF ONE DIMENSIONAL ARRAY

The process of specifying the data type, name and size of an array is called array declaration. The syntax of declaring one-dimensional array is as follows:

e.g. $\begin{array}{ccc} \text{data_type} & \text{name of array} & \text{[size];} \\ \uparrow & \uparrow & \uparrow \\ \text{int} & \text{marks} & [10]; \end{array}$

In the above example int specifies the data type of an array, marks is the name of array and [10] specifies the length of array.

INITIALIZATION OF ONE DIMENSIONAL ARRAY

The process of assigning values to array elements at the time of array declaration is called array initialization. There are different ways of initializing one dimensional array as follows:

- 1- We can initialize an array immediately after it has been declared as follows:

```
int temp[5]={2,1,-1,3,0};
```

If we provide less values than the size of array then the remaining elements are initialized with zero. The above statement is equal to following five statements.

```
temp[0]=2;    \ in this statement 2 is stored in the 1st location of array
temp[1]=1;    \ in this statement 1 is stored in the 2nd location of array
temp[2]=-1;   \ in this statement -1 is stored in the 3rd location of array
temp[3]=3;    \ in this statement 3 is stored in the 4th location of array
temp[4]=0;    \ in this statement 0 is stored in the 5th location of array
```

- 2- We can also declare and initialize an array without giving the size of array. Such an array is called un-sized array. In this case, compiler counts the number of values in curly brackets and determines the size of array like:

```
int temp[]={2,3,5,6,4}; \ Here the size is 5 because 5 values are provided in the curly bracket.
```

Program that reads 5 integer from the user and stores them in an array. It then displays all the values in the array without using loops.

```
#include<iostream>
using namespace std;
int main()
{
    int a[5];
    cout<< "Enter five values \n";
    cin>>a[0];
    cin>>a[1];
    cin>>a[2];
    cin>>a[3];
    cin>>a[4];
    cout<< "The values you entered are:\n";
    cout<<a[0]<<endl;
    cout<<a[1]<<endl;
    cout<<a[2]<<endl;
    cout<<a[3]<<endl;
    cout<<a[4]<<endl;
    return 0; }
```

Output:

```
Enter five values
6
77
85
45
90
The values you entered are:
6
77
85
45
90
```

Program that reads 5 integer values in array named 'a', and find their total and average.

```
#include<iostream>
using namespace std;
int main()
{
    int a[5],sum=0,i;
    float avg;
    for(i=0;i<=4;i++)
    {
        cout<< "Enter values: ";
        cin>>a[i];
    }
    for(i=0;i<=4;i++)
    {
        sum=sum+a[i];
    }
    avg=sum/5;
    cout<< "\n\n The total is"<<sum;
    cout<< "\n The average is"<<avg;
    return 0;
}
```

Output:

```
Enter values: 6
Enter values: 77
Enter values: 85
Enter values: 45
Enter values: 90

The total is: 303
The average is: 60
```

Program that reads 10 integer and print the biggest values in array.

```
#include<iostream>
using namespace std;
int main()
{
    int a[10],max,i;
    float avg;
    for(i=0;i<=9;i++)
    {
        cout<< "Enter values: ";
        cin>>a[i];
    }
    max=a[0];
    for(i=0;i<=9;i++)
    {
        if(a[i]>max)
            max=a[i];
    }
    cout<< "\n\n The biggest number is: "<<max;
    return 0;
}
```

Output:

```
Enter values: 60
Enter values: 77
Enter values: 85
Enter values: 45
Enter values: 190
Enter values: 31
Enter values: 110
Enter values: 490
Enter values: 20
Enter values: 322

The biggest number is: 490
```

Program that inputs 10 integer numbers in an array. The program then ask the user to enter number to search in the array. It finds the number and also the location of the number. It prints "Not Found" if the number is not in the list.

```

#include<iostream>
using namespace std;
int main()
{
    int list[10], number, i, location, flag=0;
    for(i=0; i<=9; i++)
    {
        cout<< "Enter Number for index"<<i<<": ";
        cin>>list[i];
    }
    cout<< "Enter number to search in the list";
    cin>>number;
    for(i=0; i<=9; i++)
    {
        if(list[i]==number)
        {
            flag=1;
            location=i;
            cout<<"\n\n" <<number<< "found at index"<<location;
            break;
        }
    }
    if(flag==0)
        cout<<number<< "\n\n is not found in the list";
    return 0;
}

```

Output:

```

Enter Number for index 0: 60
Enter Number for index 1: 77
Enter Number for index 2: 85
Enter Number for index 3: 45
Enter Number for index 4: 19
Enter Number for index 5: 31
Enter Number for index 6: 10
Enter Number for index 7: 90
Enter Number for index 8: 20
Enter Number for index 9: 22
Enter number to search in list: 31

31 is found at index number 5

```

Program that inputs 5 integer numbers sort the list using exchange sort / bubble sort method.

```

#include<iostream>
using namespace std;
int main()
{
    int list[5], p, c, temp;
    for(p=0; p<=4; p++)
    {
        cout<< "Enter Number : ";
        cin>>list[p];
    }
    for(p=0; p<=4; p++)
    {
        for(c=0; c<9-p; c++)
        {
            if(list[c+1]<list[c])
            {
                temp=list[c+1];
                list[c+1]=list[c];

```

Output:

```

Enter Number: 88
Enter Number: 41
Enter Number: 65
Enter Number: 10
Enter Number: 37

AFTER SORTING
10
37
41
65
88

```

```

        list[c]=temp;
    }
}
cout<< "\nAFTER SORTING\n";
for(p=0;p<=4;p++)
    cout<<"\t"<<list[p]<<endl;
return 0;
}

```

The sizeof() function

The sizeof() function is used to find the number of bytes occupied by a variable or data type. The name of the variable or data-type is given in the parenthesis.

Example program that displays the sizes of different data-types and variables.

```

#include<iostream>
using namespace std;
int main()
{
    int arr[10],b;
    cout<<"char = "<<sizeof(char)<<endl;
    cout<<"int = "<<sizeof(int)<<endl;
    cout<<"float = "<<sizeof(float)<<endl;
    cout<<"long = "<<sizeof(long)<<endl;
    cout<<"double = "<<sizeof(double)<<endl;
    cout<<"long double = "<<sizeof(long double)<<endl<<endl;
    cout<<"Variable b = "<<sizeof(b)<<endl;
    cout<<"Variable arr = "<<sizeof(arr)<<endl;
    return 0;
}

```

Output:

```

char = 1
int = 4
float = 4
long = 4
double = 8
long double = 10

```

```

Variable b = 4
Variable arr = 40

```

2-TWO DIMENSIONAL ARRAY

Two-dimensional array is also known as table or matrix that consists of rows and columns. It has two dimensions that are vertical and horizontal. The vertical dimension represents rows and the horizontal dimension represents columns. Each element in 2-D array is referred/accessed with the help of two indexes. One index is used to represent the row and the second index represents the columns of the element.

Two subscript / index values are required to reference an element of 2D array.

DECLARATION / DEFINITION OF TWO DIMENSIONAL ARRAY

data type name of array[rows][cols];
 e.g int marks[4][3]

In the above statement it declares a two-dimensional array. The first index indicated array contains four rows. The index of first row is 0 and the index of last row is 3. The second index indicates that each row in the array contains three columns. The index of first columns is 0 and the index of last column is 2. The total number of elements/cells can be determined

by multiplying rows and columns. It means the above array contains twelve elements/cells.

	marks		
	Column 0	Column 1	Column 2
Row 0	marks[0][0]	marks[0][1]	marks[0][2]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]
Row 3	marks[3][0]	marks[3][1]	marks[3][2]

INITIALIZATION OF TWO DIMENSIONAL ARRAY

We can initialize two dimensional array by using following methods:

- 1- `int list[3][3]={10,20,60,70,100,60,70,80,90};` here we provide all the values at once for all the rows.
- 3- `int list[3][3]={ {10,20,60}, {70,100,60}, {70,80,90} };` here we provide values for each row.

	list		
	Column 0	Column 1	Column 2
Row 0	10	20	60
Row 1	70	100	60
Row 2	70	80	90

Program that declares and initialize two dimension integer array of 3 rows and 4 columns, and then displays its values.

```
#include<iostream>
using namespace std;
int main()
{
    int i, j, k[3][4]={{30,20,55,206},
                      {78,81,25,90},
                      {3,48,67,104}};
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout<<"k["<<i<<"["<<j<<"]= "<<k[i][j]<<"\t";
        }
    }
    return 0;
}
```

Output:

```
k[0][0]=30 k[0][1]=20 k[0][2]=55 k[0][3]=206
k[1][0]=78 k[1][1]=81 k[1][2]=25 k[1][3]=90
k[2][0]=3 k[2][1]=48 k[2][2]=67 k[2][3]=104
```

Program that declares two dimension integer array of 3 rows and 4 columns, initialize it with some data and finds the total of all the values.

```
#include<iostream>
using namespace std;
int main()
{
    int i,j,tot, k[3][4]={ {30,20,55,206},
                          {78,81,25,90},
                          {3,48,67,104}};
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            tot=tot+k[i][j];
        }
    }
    cout<< "Total="<<total;
    return 0;
}
```

Output:

Total=807

Program that multiplies each element of an array by 2 that has 3 rows and 4 columns. It displays the values in the form of matrix. The array is initialized with the values from 1 to 12.

```
#include<iostream>
using namespace std;
int main()
{
    int i,j,k[3][4]={ {1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
    for(i=0;i<3;i++)
    {
        for(j=0;j<4;j++)
        {
            cout<< k[i][j]*2<<"\t";
        }
        cout<<endl;
    }
    return 0;
}
```

Output:

2	4	6	8
10	12	14	16
18	20	22	24

Program finds the sum of all positive numbers in the given array arr[3][4]={ {4,18,-16,11}, {-5,10,-2,12}, {15,-3,17,18}}.

```
#include<iostream>
using namespace std;
int main()
{
    int i,j,sum=0, arr[3][4]={ {4,18,-16,11}, {-5,10,-2,12}, {15,-3,17,18}};
```

```

for(i=0;i<3;i++)
{
    for(j=0;j<4;j++)
    {
        if(arr[i][j]>0)
            sum=sum+arr[i][j];
    }
}
cout<< "The sum of positive numbers = "<<sum;
}
return 0;
}

```

Output:

2	4	6	8
10	12	14	16
18	20	22	24

Program that inputs numbers in a 2 dimensional array that has r rows and c columns and print the sum of each.

```

#include<iostream>
using namespace std;
int main()
{
    int r,c,sum=0, arr[3][4];
    for(r=0;r<3;r++)
    {
        for(c=0;c<4;c++)
        {
            cout<< "Enter number";
            cin>>arr[r][c];
        }
    }
    for(r=0;r<3;r++)
    {
        sum=0;
        for(c=0;c<4;c++)
        {
            sum=sum+arr[r][c];
        }
        cout<< "Sum of row"<<r+1<<"="<<sum;
    }
    return 0;
}

```

Output:

```

Enter number: 1
Enter number: 2
Enter number: 3
Enter number: 4
Enter number: 5
Enter number: 6
Enter number: 7
Enter number: 8
Enter number: 9
Enter number: 10
Enter number: 11
Enter number: 11

Sum of row 1 = 10
Sum of row 1 = 26
Sum of row 1 = 42

```

STRING

A sequence or collection of characters is known as string. String is used to store fields like name, father name, address etc. All the string terminated with a Null character that is represented by '\0'. It means if we create a string of size 20, we can only store maximum 19 characters in it. A string in C++ is stored in one-dimensional array of characters. Each element of array stores one character. The null character is automatically appended at the end of the string.

DECLARATION / DEFINITION OF STRING

To define/declare a string in C++, the data type `char`, the name of string and the size (number of characters) that is required to store is mentioned in the declaration statement. It is important to note that the size of the string should be long enough to store null character. For example if the string value has 15 characters then the size of string array must be at least 16

Syntax: `char arrayname[size];`

e.g `char address[30];`

INITIALIZATION OF STRING

String can also be initialized same as arrays of integer and floating-point numbers in the declaration statement. For example following are the different ways to initialize a string:

- `char name[20]={"Faisal Manzoor"};`
- `char name[20]="Faisal Manzoor";`
- `char name[20]={'F','a','i','s','a','l',' ','M','a','n','z','o','o','r'};`
- `char name[]="Faisal Manzoor";` *//in this statement name variable will hold 16 characters along with a space, the null character is automatically appended to the end of the string which makes it a string of size 17.*

STRING FUNCTIONS

String functions are used to manipulate the contents of string. String functions are defined in the library file `string.h`. It means we have to use `#include<string.h>` in order to use string functions in the program. Following are the commonly used string functions in C++:

1- `cin.get()` or `cin.getline()`

It is an input function that is used to get multiword string from keyboard that may contain blank spaces in it. The general form of `cin.get()` is as follows:

`cin.get(name_of_string, max_size);` OR `cin.getline(name_of_string, max_size);`

Program that inputs name of the user using `cin.get()` function and displays it on the screen.

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
    char name[50];
    cout<< "Enter your name: ";
    cin.get(name,50); // here we can also use cin.getline(name,50); as well
    cout<< "Your name is"<<name;
    return 0;
}
```

Output:

```
Enter you name: Faisal Manzoor
Your name is Faisal Manzoor
```

2- strcpy()

It stands for string copy and is used to copy the contents of a source string variable or constant to another destination string including null character. The contents of destination string are overwritten. The general form of strcpy() function is as follows:

strcpy(destination_string_variable,source_string_variable/string_constant);

Program that initializes a string, copies it to second string and displays both the string.

```
#include<iostream>
#include<string.h>
using namespace std;

int main()
{
    char str1[30]="Karachi",str2[30];
    strcpy(str2,str1);
    cout<< "String 1 ="<<str1<<endl;
    cout<< "String 2 ="<<str2<<endl;
    return 0;
}
```

Output:

```
String 1 =Karachi
String 2 =Karachi
```

3- strcat()

It stands for string concatenation that combines the contents of source string to destination string. The contents of source string are added at the last of destination string. The general form of strcat() function is as follows:

strcat(destination string variable, source string variable);

Program that initializes a string, concatenates it to second string and displays both the string.

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
    char str1[20]="Pakistan ";
    char str2[20]="Zindabad";
    strcat(str1,str2);
    cout<< "String 1 ="<<str1<<endl;
    return 0;
}
```

Output:

```
String 1 =Pakistan Zindabad
```

4- strlen()

This function is used to find the length of a string. The length includes all characters as well as spaces in the string. It does not count the null character. The general form of strlen() function is as follows

strlen(string_variable / String_constant);

Program that initializes a string, concatenates it to second string and displays both the string with their length.

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
    char str1[20]="Pakistan ";
    char str2[20]="Zindabad";
    char str3[20]= "Pakistan Zindabad";
    cout<< "The length of String 1 ="<<strlen(str1)<<endl;
    cout<< "The length of String 2 ="<<strlen(str2)<<endl;
    cout<< "The length of String 3 ="<<strlen(str3)<<endl;
    return 0;
}
```

Output:

```
The length of String 1 =8
The length of String 1 =8
The length of String 1 =17
```

5- strcmp()

It stands for string compare that is used to compare the contents of one string to other. The string is compared character by character to other string. The comparison is case sensitive means capital A is different from small a. The comparison is based on ASCII codes of characters. The general form of strcmp() function is as follows:

strcmp(string1,string2);

The strcmp() function returns the following values depending on the arguments.

- less than zero when string 1 is less than string 2 i.e the ascii code of string 1 is less than the ascii code of string 2. e.g strcmp("ABC", "abc");
- zero when string 1 is identical to string 2 i.e the ascii code of string 1 and string 2 is same. e.g strcmp("ABC", "ABC");
- greater than zero when string 1 is greater than string 2 i.e the ascii code of string 1 is greater than the ascii code of string 2. e.g strcmp("abc", "ABC");

Program that initializes four strings and compares the contents of strings

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
    char str1[50]="KARACHI ";
    char str2[50]="QUETTA";
    char str3[50]= "LAHORE";
    char str4[50]= "KARACHI";
    int C1,C2,C3;
    C1=strcmp(str1,str2);
    cout<< "String 1 is less than String 2, so it returns "<<C1<<endl;
    C2=strcmp(str1,str4);
    cout<< "String 1 is equal to string 4, so it returns "<<C2<<endl;
```

Output:

```
String 1 is less than String 2, so it returns -1
String 1 is equal to String 4, so it returns 0
String 1 is greater than String 2, so it returns 1
```

```

C3=strcmp(str2,str4);
cout<< "String 2 is greater than string 4, so it returns "<<C3<<endl;
return 0;
}

```

6- strrev()

This function is used to reverse the contents of a string. The general form of strrev() function is as follows:

strrev(string_variable);

Program that initializes a string and displays it in reverse order.

```

#include<iostream>
#include<string.h>
using namespace std;
int main()
{
    char str1[50]="KARACHI ";
    cout<< "Str1 in proper order: "<<str1<<endl;
    cout<< "Str1 in reverse order: "<<strrev(str1);
    return 0;
}

```

Output:

```

Str 1 in proper order: Karachi
Str 1 in proper order: ihcaraK

```

EXERCISE

Q1. Select the best answer for the following MCQs.

- Which of the following is correct declaration of array?
a) int att; b) int arr{10}; c) int arr[10]; d) int arr(10);
- What is the index number of the last element of an array with 5 elements?
a) 5 b) 4 c) 0 d) 6
- What is an array?
a) An array is a series of elements
b) An array is a series of elements of same type placed in noncontiguous memory locations.
c) An array is a series of elements of the same type placed in contiguous memory locations.
d) None of the above
- Which of the following identifies the first element in array named temp?
a) temp[0] b) temp[1] c) temp(1) d) temp(0)
- Which of the following identifies the last element of array declared as int a[10][10]?
a) a[10][10] b) a[9][10] c) a[11][11] d) a[9][9]
- Given the following: int k[3][5]={ {3,10,12,27,12}, {21,20,18,25,1}, {15,16,17,44,4} }; what is in k[1][3]?
a) 12 b) 18 c) 25 d) 15
- Given the following: int arr[3][4]={ {12,0,5,10}, {7,8,19,30}, {33,1,2,22} }; which of the following statements will replace 19 with 50?
a) arr[2][3]=50; b) arr[1][2]=50; c) arr[2][1]=50; d) arr[19]=50;

8. Which of the following functions is used to append a string onto the end of another string?
a) strcpy() b) strcat() c) strlen() d) strcmp()
9. In the declaration statement `float height[8][10]`, the total number of elements will be:
a) 80 b) 18 c) 81 d) 19
10. The _____ function provides the number of bytes occupied by the data type given in the parenthesis:
a) sizeof() b) cap() c) sqrt() d) pow()
11. A string constant is written in:
a) Single quotes b) Double quotes c) Without quotes d) Parenthesis
12. All strings end with a special character called NULL character that is represented by:
a) \n b) \e c) \0 d) 0
13. If `char city[] = "Karachi";` then what will be the size of the string variable city?
a) 8 b) 7 c) 9 d) None
14. The `strcmp()` function will return _____ if string 1 and string 2 are same:
a) 0 b) 1 c) -1 d) None
15. The _____ function is used to return the length of a string.
a) strlen b) strlengh c) len d) strsize

Q2. Write short answers of the following questions.

1. Define array and give its advantages in programming.

Ans. See Ans on Page 80

2. Differentiate between one dimensional and two dimensional Array.

Ans. One dimensional array consists of only single row or single column. Each element of this array can be accessed by using single index value where as two dimensional array consists of multiple rows and columns. Each element of array is accessed using two index values, one for the row and one for the column.

3. What is the purpose of `sizeof()` function? Give one example.

Ans. See Ans on Page 87

4. Declare an array names x, that has 3 rows and 5 columns and assign it values from 1 to 15 in the declaration statement.

Ans. `int x[3][5] = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};`

5. Define string and explain how it is stored in computer memory.

Ans. A sequence of characters written in double quotations is called a string. It is used to represent field like name, address, book title etc. A string in C++ is stored in one dimensional array of characters. Each element of array stores one character. All the strings end with a special character, known as null character and it is represented by '\0'. The null character automatically appended at the end of the string.

6. What is the advantage of using `cin.get()` function over `cin` statement for reading a string.

Ans. The advantage of using `cin.get` function over `cin` statement is that `cin` statement considers a space as terminating character where as `cin.get()` function considers enter key as a terminating character

therefore we can input multiword string using `cin.get()` and with `cin` statement we cannot enter string consisting of space.

7. Differentiate between `strcpy()` and `strcmp()` functions.

Ans. The `strcpy()` function is used to copy the contents a string variable or string constant to another string variable. The general form of `strcpy()` function is: `strcpy(string1, string2)`
The `strcmp()` function compares two strings and returns an integer values based on the result of comparison. This comparison is based on ASCII codes of characters. The general form of `strcmp()` function is: `strcmp(string1, string2)`

- It will return 0 if both string are same.
- It will return 1 if string1 is greater than sting2.
- It will return -1 if string1 is less than sting2.

8. Differentiate between `strlen()` and `strcat()` functions.

Ans. The `strlen()` function is used to find the length of a string. The length includes all characters as well as spaces in the string but the null character is not included. The general form of `strlen()` function is: `strlen(string)`

The `strcat()` function is used to append/join the one string at the end of other string. The general form of `strcat` function is: `strcat(string1, string2)`

Q3. Write long answers of the following questions.

1. What is array? Explain one dimensional array in detail with one example.

Ans. **Array:**

An array is used to store set of values of same data types. It is a collection of consecutive memory locations with same name and type. Each memory location of an array is called element of array. The total number of elements in the array is called its length. Array can store huge amount of data with single variable name. Array is used to process a list of numbers and strings easily and quickly. Various operations can be performed on the values stored in an array such as searching and sorting etc.

Each element in the array is accesses with reference to its position or location in the array. The position is called index or subscript. In C++ the index of an array always start with 0 therefore the index of last element is less than the size of the array.

One-Dimensional Array

A type of array in which all elements are arranged in the form of list is known as one-dimensional array. It consists of only one column and uses single subscript number to access the element.

Declaration of One-Dimensional Array

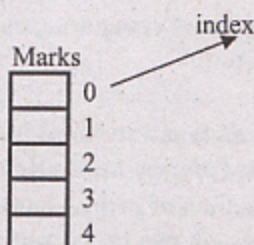
The syntax of declaring one-dimensional array is as follows:

`datatype arrayname[size];`

Example:

`int marks[5];`

The above example declares an integer array marks with five elements. It allocates five consecutive locations in memory. The index of its first element is 0 and the index of its last element is 4.



Initialization of One-Dimensional Array

The syntax of Initializing one-dimensional array is as follows:

datatype arrayname[size]={values};

Example:

int marks[5]={61,83,91,77,60};

The above example declares an integer array marks with five elements. It also initializes the array with values given in the braces. 61 is stored in marks[0], 83 is stored in marks[1] and so on.

	0	1	2	3	4
marks	61	83	91	77	60

2. Explain two dimensional array in detail with one example.

Ans. See Ans on Page 84

3. What are strings? How strings are defined in C++? Give examples.

Ans. See on Page 87,88

4. Explain the purpose of the following string functions.

i) strcpy() See Ans on Page 89

ii) strcat() See Ans on Page 89

iii) strlen() See Ans on Page 89

iv) strcmp() See Ans on Page 90

5. Determine the output of the following codes:

1) int a[5]={10,3,5,1,2}; for(int i=4;i>0;i--) { a[i]=a[i-1]; cout<<a[i]<<" "; } 	2) int a[5]={4,3,6,8,7}; for(int i=0;i<=4;i+=2) { a[i]=2*a[i]; cout<<a[i]<<" "; }
3) int a[5]; a[4]=3; for(int i=3;i>=0;i--) { a[i]=2*a[i+1]-i; cout<<a[i]; } 	4) int i,j,m; int a[5]={8,10,1,14,16}; i=++a[2]; m=a[i++]; cout<<i<<m;
5) int arr[5]={12,43,36,8,7},c,s=0; for(c=0;c<=4;c++) { if(arr[c]%3==0) { 	6) int b[4]={4,5,2,1}; for(int i=2;i>=0;i--) cout<<b[i]+b[i+1];

<pre>cout<<arr[c]<<"\n"; s+=arr[c]; } } cout<<"Sum="<<s;</pre>	
<pre>7) char ch[3]={ 'a', 'b', 'c'}; for(index=0;index<3;index++) cout<<ch[index];</pre>	<pre>8) char name[20]="Computer"; for(int i=strlen(name);i>=0;i--) cout<<name[i];</pre>

Ans.

1) Ans. 3 6 8 13	2) Ans. 8 12 14
3) Ans. 34714	4) Ans. 32
5) Ans. 12 36 Sum = 48	6) Ans. 379
7) Ans. abc	8) Ans. retupmoC

Lab Activities / Programs

1. Write a program that reads ten numbers in an array and prints them in reverse order.

Ans.

```
#include<iostream>
using namespace std;
int main()
{
int a[10],i;
cout<<"Enter ten numbers\n";
for(i=0;i<=9;i++)
cin>>a[i];
cout<<"\nNumbers in reverse order\n";
for(i=9;i>=0;i--)
cout<<a[i]<<endl;
return 0;
}
```

2. Write a program that reads ten numbers and print the smallest along with its index.

Ans.

```
#include<iostream>
using namespace std;
int main()
{
int a[10],i,smallest,index;
cout<<"Enter ten numbers\n";
for(i=0;i<=9;i++)
cin>>a[i];
```

```

smallest=a[0];
for(i=9;i>=0;i--)
{
    if(a[i]<smallest)
    {
        smallest=a[i];
        index=i;
    }
}
cout<<"\nThe smallest number is "<<smallest<<endl;
cout<<"\n The index is "<<index;
return 0;
}

```

3. For the given array: `int arr[15]={4,8,1,1,5,0,12,5,7,3,15,8,4,11};`
Write a program that prints the number of times the number 5 appears in the array.

Ans. `#include<iostream>`
`using namespace std;`
`int main()`
`{`
`int arr[15]={4,8,1,1,5,0,12,5,7,3,15,8,4,11},count=0,i;`

`for(i=0;i<=14;i++)`
`{`
`if(arr[i]==5)`
`count=count+1;`
`}`
`cout<<"\nThe 5 appears "<<count<<" times in the array";`
`return 0;`
`}`

4. For the given array: `int a[3][2]={{6,3},{7,8},{4,5}};`
Write a program that displays all the elements in the form of matrix as shown below and finds its sum.

6	3
7	8
4	5

The sum is : 33

Ans. `#include<iostream>`
`using namespace std;`
`int main()`
`{`
`int a[3][2]={{6,3},{7,8},{4,5}},i,j,sum=0;`
`for(i=0;i<=2;i++)`
`{`
`for(j=0;j<=1;j++)`

```

{
    cout<<a[i][j]<<"\t";
    cout<<endl;
    sum=sum+a[i][j];
}
}
cout<<"\n\nThe sum is "<<sum;
return 0;
}

```

5. For the given array: `int arr[3][4]={{4,18,-16,11},{-5,10,-2,12},{15,-3,17,18}};`
Write a program to find the sum of positive numbers.

Ans. `#include<iostream>`
`using namespace std;`
`int main()`
`{`
`int arr[3][4]={{4,18,-16,11},{-5,10,-2,12},{15,-3,17,18}},i,j,sum=0;`
`for(i=0;i<=2;i++)`
`{`
`for(j=0;j<=3;j++)`
`{`
`if(arr[i][j]>0)`
`sum=sum+arr[i][j];`
`}`
`}`
`cout<<"\n\nThe sum of positive number is "<<sum;`
`return 0;`
`}`

6. For the array: `int a[2][4]={{14,8,11,10},{15,12,20,3}},b[2][4]={{2,3,4,7},{6,7,8,9}};`
Write a program that adds the two array and produce the following output.

Sum of two arrays is:

15	11	15	17
21	19	28	12

Ans. `#include<iostream>`
`using namespace std;`
`int main()`
`{`
`int a[2][4]={{14,8,11,10},{15,12,20,3}},b[2][4]={{2,3,4,7},{6,7,8,9}},c[2][4],i,j;`
`for(i=0;i<2;i++)`
`{`
`for(j=0;j<=3;j++)`
`{`
`c[i][j]=a[i][j]+b[i][j];`
`}`
`}`
`}`

```

cout<<"\nThe sum of two array is:\n";
for(i=0;i<2;i++)
{
    for(j=0;j<=3;j++)
    {
        cout<<c[i][j]<<"\t";
    }
    cout<<endl;
}
return 0;
}

```

7. Input data from keyboard in a two dimensional array x, that has r rows and c columns and print the sum of each row in the following format.

Sum of row 1 =

Sum of row 2 =

Sum of row 3 =

.

.

.

Sum of row r =

Ans. #include<iostream>
using namespace std;
int main()
{
int x[3][3],r,c,sum;

```

for(r=0;r<3;r++)
{
    for(c=0;c<3;c++)
    {
        cout<<"Enter values \n";
        cin>>x[r][c];
    }
}
for(r=0;r<3;r++)
{
    sum=0;
    for(c=0;c<3;c++)
    {
        sum=sum+x[r][c];
    }
    cout<<"\nThe sum of row"<<r+1<<"="<<sum;
}
return 0;
}

```

8. Write a program that reads a string, copies it into another string and then prints both the strings.

Ans. See Ans on Page 89

9. Write a program that reads 2 strings of size 20 and perform the following operations:
 a) Print both strings with their length.
 b) Concatenate the second string onto the end of first string and print it.

Ans.

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
char str1[40], str2[20];
cout<<"Enter 1st string";
cin.getline(str1,40); //cin.get() has some built-in problem thats why cin.getline is used
cout<<"Enter 2nd string";
cin.getline(str2,20); //cin.get() has some builti-n problem thats why cin.getline is used
cout<<"The length of String 1 ="<<strlen(str1)<<endl;
cout<<"\n The length of String 2 ="<<strlen(str2)<<endl;
cout<<"\n First String Concatenated into Second string "<<strcat(str1,str2);
return 0;
}
```

10. Write a program that reads 3 strings and prints the smallest.

Ans.

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
char str1[20], str2[20],str3[20];
cout<<"Enter 1st string";
cin.getline(str1,20); //cin.get() has some builtin problem thats why cin.getline is used
cout<<"Enter 2nd string";
cin.getline(str2,20); //cin.get() has some builtin problem thats why cin.getline is used
cout<<"Enter 3rd string";
cin.getline(str3,20); //cin.get() has some builtin problem thats why cin.getline is used
if(strlen(str1)<strlen(str2) && strlen(str1)<strlen(str3))
cout<<"\nString 1 is smallest";
else if(strlen(str2)<strlen(str1) && strlen(str2)<strlen(str3))
cout<<"\nString 2 is smallest";
else
cout<<"\nString 3 is smallest";
return 0;
}
```

UNIT 6

FUNCTIONS

FUNCTION

Function is a group of statements that is written to perform a specific task. Functions are building blocks of C++ programs. Every C++ program has at least one function, which is known as `main()` function. Additional functions are used as required by the program.

ADVANTAGES OF USING FUNCTION

- 1- Creating applications using function makes it easier to understand, edit, and debug the program.
- 2- Functions are written only once but may be referenced many times at different points in the program so to avoid unnecessary duplication of code.
- 3- If we break lengthy program into smaller logical blocks then it is easier to write small codes instead of writing a long program. A programmer can focus the attention on a specific code.
- 4- Each function has a unique name and is written as an independent block. If there is an error in the program, the change is made to the particular function in which the error exists.
- 5- The code written in functions can be reused as and when required, meaning one function can be used in many programs.

TYPES OF FUNCTION

- 1- Library Functions / built-in Functions
- 2- User defined Functions.

1- Library Functions / Built-in Functions

Library functions are the part of C++ that are already defined in different header files such as `math.h`, `conio.h`, `string.h` etc. `sqrt()`, `getch()`, `strcmp()` etc. are some of the examples of Library function. Library functions are also known as built-in functions or pre-defined functions.

2- User defined Functions

Functions that are created by the users according to their program requirements are called user-defined functions.

Any function that is user defined or pre-defined function must belong to the following category of function.

- i) A function that can neither accept values nor return values. e.g.
`void function_name(void)`
- ii) A function that can accept values but doesn't return values. e.g.
`void function_name(parameters)`
- iii) A function that cannot accept values but can return values. e.g.
`return-type function_name (void)`
- iv) A function that can accept values as well as return values. e.g.
`return-type function_name (parameters)`

COMPONENTS / PARTS OF FUNCTION

There are three parts of function as:

1- Function Prototype (Function Declaration) 2- Function Call 3- Function Definition

1- FUNCTION PROTOTYPE / DECLARATION / FUNCTION SIGNATURE

Function declaration or prototype is very similar to variable declaration. Functions must be declared before they are used. Function declaration tells 3 things to the compiler as:

i) The name of function ii) The data-type the function returns, if any. iii) The number and data-types of function's parameters if any. For example:

void function_name(void);

First void before the function name means that the function does not return any value and the second void in the brackets means it doesn't accept any value. More over the function declaration always end with semicolon.

2- FUNCTION CALL

Function call causes the control to be transferred to the function definition where actual code of function is written. After executing the statements in the body of function, the control returns to the calling function following the function call statement.

3- FUNCTION DEFINITION

Function definition is an actual code of the function that consists of statements or set of statements that are written to perform the specific task. It starts with a line which is very similar to function declaration. It tells the compiler that a function is being defined. The function heading doesn't end with semicolon. Function definition can be written before or after main() function but it is always outside the main function. If function definition is defined before the main() function then there is no need of function declaration. Function definition consists of two parts:

i) Function Header

It is also called decelerator and the first line of function definition. It is same as the function declaration but it is not terminated by semicolon (;).

ii) Function Body

The statements that are written with in the braces of function to perform specific task is known as body of the function.

EXAMPLE PROGRAM TO EXPLAIN THE COMPONENTS OF FUNCTION

```

void line(void);  → Function declaration, ends with semicolon
{
    cout<< "I love Pakistan\n";
    line();
    cout<< "\n It is very beautiful country\n";
    line();
}
void line(void) → Header doesn't end with semicolon
{
    cout<< "===== "; → Body of Function
}

```

Diagram annotations:

- Function Call:** An arrow points from the `line();` calls inside the first function to the `void line(void)` definition.
- Calling Function:** A bracket groups the first function (the one that calls `line()`).
- Function Definition:** A bracket groups the second function (the one that is called).

SCOPE AND LIFE TIME OF VARIABLE

The area/ part of program where a variable can be accessed or used is known as scope of variable whereas the time period for which a variable exists in the memory is known as lifetime of a variable. It is the time of the variable creation to variable destruction from memory. There are three types of variables with respect of scope and life time as follows:

- 1) Local scope variable
- 2) Global scope variable
- 3) Static scope variable

1- LOCAL SCOPE VARIABLE / AUTOMATIC VARIABLE

Local variable can be called as Private variable, or Automatic variable. Local variables are those variables that are visible or accessible only within the block/function in which they are defined and cannot be used outside of that block/function.

Local variables are created when the function containing them is called and destroyed when the function terminates. The initial value of local variable is garbage.

Example program to understand the concept of local variable.

```
#include<iostream>
```

```
void test(void);
```

```
int main()
```

```
{
```

```
    int a;      // variable a is local variable because it is declared inside main function
```

```
    a=10;
```

```
    cout<< "The value of a is"<<a;
```

```
    test();
```

```
    //cout<< "The value of b is" <<b; this is invalid statement because variable 'b' is defined
    //in function test() and cannot be used in main() function
```

```
}
```

```
void test(void)
{
    int b;      // variable b is local variable because it is declared inside fun() function
    b=20;
    cout<< "The value of b is"<<b;
    //cout<< "The value of a is" <<a;  this is invalid statement because variable 'a' is defined
                                     //in function main() and cannot be used in fun() function
}
```

2- GLOBAL SCOPE VARIABLE / PUBLIC VARIABLE

Global variable can be called as Public variable. Global variables are those variables that are visible or accessible to all the blocks/functions in the program. These variables are defined outside of any function/block.

Global variables exist in the memory as long as the program is running and destroyed when the program terminates. The initial value of global variable is 0.

Example program to understand the concept of global variable.

```
#include<iostream>
```

```
int a;      // variable 'a' is Global variable because it is declared outside of any function
void test(void);
int main()
{
    a=10;      // variable 'a' can be used in main() function
    test();
}
void test(void)
{
    cout<< "The value of a is" <<a;  //this is valid statement because variable 'a' is defined
                                     //outside of any function and can be used in any function.
}
```

3- STATIC VARIABLE

A local variable declared with keyword 'static' is called static variable. The scope of static variable is similar to the scope of local variable and the life time of static variable is similar to the life time of global variable.

The static variables are created in the memory when the function is called for the first time but are not destroyed when the control exits from the function. These variables exist in memory as long as the program is running and destroyed when the program terminates. The static variable preserve the last value of a function returned.

Example program to understand the concept of static variable.

```
#include<iostream>
using namespace std;
void test(void);
int main()
{
    int i;
    for(i=1;i<=3;i++)
    {
        test();
    }
    return 0;
}
void test(void)
{
    int c1;
    static int c2=0;
    cout<< "The value of local variable c1 is\n"<<c1;
    cout<< "The value of static variable c2 is"<<c2;
    c1++;
    c2++;
}
```

Output:

```
The value of local variable is:0
The value of Static variable is:0
The value of local variable is:0
The value of Static variable is:1
The value of local variable is:0
The value of Static variable is:2
```

Example program that passes two values as argument and print the sum and product using two separate functions.

```
void sum(int n1,int n2);
void prod(int x,int y);
#include<iostream>
using namespace std;
int main(void)
{
    int num1,num2;
    clrscr();
    cout<<"Enter 1st Number ";
    cin>>num1;
    cout<<"Enter 2nd Number ";
    cin>>num2;
    sum(num1,num2);
    prod(num1,num2);
    return 0;
}
void sum(int n1,int n2)
```

Output:

```
Enter 1st Number 5
Enter 2nd Number 10
The Sum of 5 and 10 is 15
The Product of 5 and 10 is 50
```

```

    cout<<"\nThe Sum of"<<n1<<"and"<<n2<<"is"<<n1+n2;
}
void prod(int x,int y)
{
    cout<<"\nThe Product of"<<n1<<"and"<<n2<<"is"<<x*y;
}

```

Example program that passes an argument and print its table in a user defined function.

Output:

```

void table(int t)
{
    int i;
    for(i=1;i<=10;i++)
        cout<<"\n"<<t<<" x "<<i<<" = "<<t*i;
}
#include<iostream>
using namespace std;
int main(void)
{
    int n1,n2;
    cout<<"Enter 1st Number ";
    cin>>n1;
    table(n1);
    cout<<"Enter 2nd Number";
    cin>>n2;
    table(n2);
    return 0;
}

```

Enter 1st Number 3

3 x 1 = 3

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12

3 x 5 = 15

3 x 6 = 18

3 x 7 = 21

3 x 8 = 24

3 x 9 = 27

3 x 10 = 30

Enter 2nd Number 5

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

RETURNING VALUE FROM FUNCTION

A function can return only one value at a time. The return type in function declaration indicates the type of value a function can return. The keyword **return** is used to return a value to the calling function. It also transfers the control back to the calling function.

Example program that passes two values as argument and returns the sum of that values using function.

```

int sum(int n1,int n2);
#include<iostream>
using namespace std;

int main(void)
{
    int num1,num2;

```

```

    cout<<"Enter 1st Number";
    cin>>num1;
    cout<<"Enter 2nd Number";
    cin>>num2;
    cout<<"\nThe sum of <<num1<< "and"<<num2<< "is"<<num1+num2;
    return 0;
}
int sum(int n1,int n2)
{
    return (n1+n2);
}

```

SCOPE OF FUNCTION

The area / part of program where a function can be accessed or used is known as scope. The scope of any function is determined by the place of function declaration. There are two types of functions with respect of scope as follows:

- 1) Local Function
- 2) Global Function

1- LOCAL FUNCTION

Local function is a type of function that is declared inside the body of another function. A local function can be called within the function in which they are defined.

Example program to understand the concept of Local Function.

```

#include<iostream>
using namespace std;
void show(void);
int main()
{
    show();
    //local(); here it is invalid statement because local function cannot be called from other function
    return 0;
}
void local(void)
{
    cout<<"Local function";
}
void show(void)
{
    void local(void); // declared inside show() function
    local(); // here it is valid statement because local function can be called from its function
}

```

2- GLOBAL FUNCTION

Global function is a type of function that is declared outside the body of another function. A global function can be called from any part of the program. The global functions are typically declared at the top of the program before main() function.

Example program to understand the concept of Global Function.

```
#include<iostream>
using namespace std;
void print(void);      // it's a global function declared outside of any function
void test(void);       // it's a global function declared outside of any function
int main()
{   print();           // can be called from main() function
    test();
    return 0; }
void test(void)
{
    print();           // can be called from test() function as well
}
void print(void)
{   cout<< "Testing global function"; }
```

INLINE FUNCTION

The function declared with the keyword **inline** before the return type is known as inline function. The purpose of inline function is to save the CPU time because a lot of CPU time is wasted in passing control from calling function (main function) to the called function and returning control back to the calling function. This limitation can be overcome by using inline function.

When we use inline function, it requests the compiler to treat the function code as an inline and do not jump again and again to the calling function and back to it. When the compiler compiles the code, all inline function codes are expanded in-place which means the function call is replaced with the copy of the contents of the function itself. It saves time and increase efficiency. The disadvantage of inline function is that the compiled code will be very lengthy. It also takes more memory than normal function call. That is why it is typically used when the function contains small piece of code.

Example program to understand the concept of inline function

```
inline int square(int num);
#include<iostream>
using namespace std;
int main(void)
{
    int n1=3,n2=4,n3=6;
    cout<<"square of"<<n1<<"="<<square(n1);
```

```

    cout<<"\n square of "<<n2<<"="<<square(n2);
    cout<<"\n square of "<<n3<<"="<<square(n3);
}
inline int square(int num)
{
    return (num*num);
}

```

DEFAULT FUNCTION

The default arguments allow the user to call a function without giving required arguments. The initialized values of default arguments are used if the user does not specify the value for the arguments in function call. To specify default arguments we simply have to use the assignment operator and a value for the arguments in function declaration like:

```
type function_name(parameter1=value,...);
```

ADVANTAGES OF DEFAULT ARGUMENT

- 1) It is easier to use as the user does not need to specify the values.
- 2) It minimizes the chance of error in providing wrong parameters.
- 3) It provides more flexibility in calling function.
- 4) It allows more consistency with the values.

DISADVANTAGES OF DEFAULT ARGUMENT

- 1) It can be confusing if default values are not used for all parameters.
- 2) It may take more time to modify a program.

25Example program to understand the concept of default argument.

```
int sum(int num1, int num2=10)
```

```

{
    int result;
    result=num1+num2;
    return result;
}
#include<iostream>
using namespace std;
int main(void)
{
    cout<<"Result by using one argument="<<sum(15);
    cout<<"\n Result by using both arguments="<<sum(8,30);
}

```

Output:

```

Result by using one argument=25
Result by using two arguments=38

```

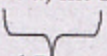
PARAMETERS

Parameters or parameter list are the values that are provided in a function prototype or when the function called. The values are written within parenthesis. There are two types

of parameters as follows:


1- Formal Parameters:

Formal parameters are those parameters which are used in function prototype and in function definition header. These parameters are used to receive values from the calling function.

e.g `void sum(int a, int b)`

 Formal Parameters

2- Actual Parameters:

Actual parameters are those parameters which are used in function call. These are the actual values that are passed to the function when it is called. The actual parameters may be in the form of constant or variables. Actual parameters are also known as arguments.

e.g `sum(20,60)`

 Actual parameters /Arguments

Example program to understand the difference between Formal parameters and Actual parameters.

```

void sum(int n1,int n2); // Formal Parameters
#include<iostream>
using namespace std;
int main(void)
{
    int num1,num2;
    clrscr();
    cout<<"Enter 1st Number ";
    cin>>num1;
    cout<<"Enter 2nd Number ";
    cin>>num2;
    sum(num1,num2); // Actual Parameters
    return 0;
}

void sum(int n1,int n2) // Formal Parameters
{
    cout<<"\nThe Sum of" <<n1<<"and"<<n2<<"is"<<n1+n2;
}

```

PASSING ARGUMENTS TO A FUNCTION

We can call a function and pass arguments to the function by the following method:

1- Function call by constant 2- Function call by value 3- Function call by reference

1- PASSING ARGUMENTS BY CONSTANT

The argument passing mechanism in which, the actual constant values (numeric or character) are passed instead of passing the variables holding these constants.

Example program to understand the concept of passing argument by constant.

```
void test(int x)
{
    cout<< "The value of x is"<<x<<endl;
}
#include<iostream>
using namespace std;
int main(void)
{
    test(10);    // The constant value 10 is passed
    return 0;
}
```

2- PASSING ARGUMENTS BY VALUE

A parameter passing mechanism in which the value of each of the actual parameter is copied into corresponding formal parameter. With this method the changes made to formal parameter does not effect on the values of actual arguments. It is the default method for passing parameters to a function in C++. Passing argument by value is useful when the function does not need to modify the original variable.

Example program to understand the concept of passing argument by variable.

```
void interchange(int x, int y);
int main(void)
{
    int n1, n2;
    n1=10;
    n2=20;
    cout<<"n n1="<<n1<<" n2="<<n2;
    interchange(n1,n2);
}
```

```
void interchange(int x, int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
    cout<<"\nx="<<x<<" y="<<y;
}
```

2- PASSING ARGUMENTS BY REFERENCE

A parameter passing mechanism in which the addresses of actual parameter in the calling function are copied into formal parameter of the called function. The formal parameter is not created separately in the memory, it becomes a second name of actual parameter. It means that single memory location is shared between actual parameter and formal parameter. The pass by reference method is useful when we want to manipulate original values. The ampersand (&) sign is used before the variables of formal parameters to indicate that the argument will be passed by reference.

Example program to understand the concept of passing argument by reference.

```
void interchange(int &x, int &y);
```

```
#include<iostream>
```

```
using namespace std;
```

```
int main(void)
```

```
{
    int n1, n2;
    n1=10;
    n2=20;
    interchange(n1,n2);
    cout<<"\n n1="<<n1;
    cout<<"\n n2="<<n2;
    return 0;
}
```

```
void interchange(int &x, int &y)
```

```
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

FUNCTION OVERLOADING

Function overloading is the process of declaring multiple functions with same name but different parameters. The function with same name must differ in one of the following ways:

- 1) Number of parameters
- 2) Type of parameters

1) Number of parameters

Functions can be overloaded if they have different numbers of parameters. More than one function with the same name but different number of parameters can be used in a single program. When the function is called, the compiler differentiates the calls by looking at the number of arguments.

Example program to understand the concept of Function Overloading.

```
int product(int a, int b)
{
    return(a*b);
}
int product(int a, int b, int c)
{
    return(a*b*c);
}
#include<iostream>
using namespace std;
int main(void)
{
    int n1=10,n2=5,n3=2;
    cout<< "Product of 2 integers="<<product(n1,n2)<<endl;
    cout<< "Product of 3 integers="<<product(n1,n2,n3);
    return 0;
}
```

2) Types of parameters

Functions can be overloaded if they have different types of parameters. More than one function with the same name but different types of parameters can be used in a single program. When the function is called, the compiler differentiates the calls by looking at the types of arguments. For example:

```
float product(float a, float b)
{
    return(a*b);
}
int product(int a, int b)
{
    return(a*b);
}
```

```
#include<iostream>
using namespace std;
int main(void)
{
    int n1=10,n2=5;
    float n3=2.5,n4=5.1;
    cout<< "Product of 2 integers="<<product(n1,n2)<<endl;
    cout<< "Product of 2 floating numbers="<<product(n3,n4);
    return 0;
}
```

RETURN TYPE IS NOT CONSIDERED AS FUNCTION OVERLOADING

Return type of a function is not considered when functions are overloaded. It means that if two or more functions with the same name have same function signatures but different return types then they are not considered as overloaded and the compiler will generate error message. For example:

```
int print();
double print();
```

The error message will be generated because it is not considered as function overloaded.

ADVANTAGES OF FUNCTION OVERLOADING

Following are the main advantages of function overloading:

- 1- The execution of program will be speeds up.
- 2- It is used to achieve polymorphism, which means that same function performs different tasks depending on the parameters.
- 3- The user can easily remember the function name because same name is used for different tasks.
- 4- It provides readability and consistency in the program.

DIFFERENC BETWEEN LOCAL VARIABLE AND GLOBAL VARIABLE.

LOCAL VARIABLE	GLOBAL VARIABLE
1) Local variable is declared with inside a function block.	1) Global variable is declared outside of any function block.
2) It can be used only inside a function in which it is declared.	2) It can be in all the functions in the program.
3) It is created when the control enters the function in which it is created.	3) It is created when the control enters the program.
4) It is destroyed when the control exits from the function in which it is created.	4) It is destroyed when the control exits from the program.
5) It is used when the values are to be used within a function.	5)-It is used when the values are to be shared among different functions.

DIFFERENC BETWEEN PASS BY VALUE AND PASS BY REFERENCE.

PASS BY VALUE	PASS BY REFERENCE
1) It passes the copy of actual parameter to formal parameter.	1) It passes the address of actual parameter to formal parameter.
2) The actual and formal parameters refer to different memory locations	2) The actual and formal parameters refer to same memory locations.
3) Any change made by the function in formal parameter does not affect the value of actual parameters.	3) Any change made by the function in formal parameter changes the value of actual parameters.
4) It requires more memory.	4) It requires less memory.
5) It does not use & sign with parameters.	5) It uses & sign with parameters.

EXERCISE

Q1. Select the best answer for the following MCQs.

1. The phenomenon of having two or more functions in a program with the same names but different number and types of parameters is known as:

- a) inline function b) nested function c) function overloading d) recursive function

2. We declare a function with _____ if it does not have any return type.

- a) long b) double c) void d) int

3. Arguments of a functions are separated with _____

- a) comma (,) b) semicolon (;) c) colon (:) d) none of these

4. Variable inside parenthesis of function declarations have _____ level access.

- a) local b) global c) module d) universal

5. Observe the following function declaration and choose the best answer.

int divide(int a, int b=2)

- a) Variable b is of integer type and will always have value 2.
 b) Variable a and b are of int type and the initial value of both variable is 2.
 c) Variable b is international scope and will have value 2.
 d) Variable b will have value 2 if not specified when calling function.

6. Function declaration consists of:

- a) Function name b) Return type
 c) Number and types of parameters d) All

7. Which of following is true about function prototype:

- a) It is a single statement b) It is also called function declaration
 c) It ends with semicolon d) All

8. Which of the following is NOT valid function declaration?

- a) int ave(int a, int b, int c); b) int 3ave(int a, int b, int c);
 c) int ave3(int, int, int); d) int ave_3(int a1, int a2, int a3);

9. Function definition can be written:

- a) Before main() function b) After main() function
 c) In a separate file d) All

10. The process of sending an argument to a function is called:

- a) Sending b) Filtering c) Delivering d) Passing

11. The scope of variable refers to:
 - a) Length of variable
 - b) Name of variable
 - c) Accessibility of variable
 - d) Data type of variable
12. The local variables are also called:
 - a) External variables
 - b) Automatic variables
 - c) Dynamic variable
 - d) All
13. The declaration `void fun(int x, int &y);` is called:
 - a) Function body
 - b) Function type
 - c) Function stereo type
 - d) Function prototype
14. The parameters are passed by reference by the symbol:
 - a) #
 - b) *
 - c) @
 - d) &
15. In function overloading, the function differ with:
 - a) Types of arguments
 - b) Number of arguments
 - c) Sequence of arguments
 - d) All

Q2. Write answers of the following questions.

1. What is function? Explain different types of functions used in C++ with examples.
 Ans. See Ans on Page 100
2. Explain function components with examples.
 Ans. See Ans on Page 101.102
3. Define default arguments. Give the advantages and disadvantages of default arguments.
 Ans. See Ans on Page 108
4. What is meant by the term function overloading? How a function can be over loaded in C++? Explain it with the help of an example program.
 Ans. See Ans on Page 112
5. What is function signature? Explain its different parts.
 Ans. Function signature is the model of a function. It provides information to compiler about the structure of the function to be used in program. It is also known as function declaration or function prototype. It comprises of the following parts:
 - i) **Name of the function:** It indicates the name of the function. Each function should have a unique name.
 - ii) **Arguments/Parameters:** Parameters are given within the parentheses after the name of the function which indicates that how many values a function will accepts from a calling function. If the function is not accepting any value then the empty parentheses are used or a keyword `void` is used.
 - iii) **Return type:** It indicates the type of values that will be returned by function.

Example to understand the signature of function.

```
double add(int x, double y)
{
    return x+y;
}
```

The signature of above add function comprises of:

- i) The name of the function is "add"
- ii) The Arguments or parameters are "int, double"
- iii) Return type is "double"

6. Explain the scope of different types of variables used in functions.

Ans. See Ans on Page

7. What are parameters? Explain their types with examples.

Ans. See Ans on Page

Lab Activities / Programs

1. Write a program with a function that takes two int parameters, adds them together, and then returns the sum.

Ans.

```
#include<iostream>
#include<string.h>
using namespace std;
int sum(int n1,int n2);
int main()
{
    int a,b,s;
    cout<<"Enter 1st value";
    cin>>a;
    cout<<"Enter 2nd value";
    cin>>b;
    s=sum(a,b);
    cout<<"The Sum is "<<s;
    return 0;
}
int sum(int n1,int n2)
{
    return n1+n2;
}
```

2. Write a program with a function name "mean" to read three integers from the keyboard to find the arithmetic mean.

Ans.

```
#include<iostream>
#include<string.h>
using namespace std;
int mean(int n1,int n2,int n3);
int main()
{
    int a,b,c;
    cout<<"Enter 1st value";
    cin>>a;
    cout<<"Enter 2nd value";
    cin>>b;
    cout<<"Enter 3rd value";
    cin>>c;
    mean(a,b,c);
    return 0;
}
```

```

}
int mean(int n1,int n2,int n3)
{
    int r;
    r=(n1+n2+n3)/3;
    cout<< "The Mean is "<<r;
}

```

3. Write a C++ program having function name rectangle to read the length and width of a rectangle from the keyboard and find area of the rectangle. The result should be returned to the main program for displaying on the screen.

Ans.

```

#include<iostream>
#include<string.h>
using namespace std;
int rectangle();
int main()
{
    int a;
    a=rectangle();
    cout<<"The Area of Rectangle is "<<a;
    return 0;
}
int rectangle()
{
    int l,w,area;
    cout<<"Enter length of a rectangle ";
    cin>>l;
    cout<<"Enter width of a rectangle ";
    cin>>w;
    area=l*w;
    return area;
}

```

4. Write a C++ program having two function names area and perimeter to find the area and perimeter of a square. (Area=lengthofsidex lengthofsidex, (perimeter=lengthofsidex x 4)

Ans.

```

#include<iostream>
#include<string.h>
using namespace std;
void area(int l);
void perimeter(int l);
int main()
{
    int length;
    cout<<"The Length of Square ";
    cin>>length;
    area(length);
    perimeter(length);
}

```

```
        return 0;    }  
void perimeter(int l)  
{  
    int p;  
    p=l*4;  
    cout<<"\nThe perimeter of square is "<<p;  
}  
void area(int l)  
{  
    int a;  
    a=l*l;  
    cout<<"\nThe area of square is "<<a;  
}
```

5. Write C++ program to read a number from the keyboard and then pass it to a function to determine whether it is prime or composite.

Ans. #include<iostream>

```
void prime(int p);  
using namespace std;  
int main()  
{  
    int num;  
    cout<<"Enter a number ";  
    cin>>num;  
    prime(num);  
    return 0;  
}  
void prime(int n)  
{  
    int i, flag=1;  
    for(i=2; i<n; i++)  
    {  
        if(n%i==0)  
        {  
            flag=0;  
            break;  
        }  
    }  
    if(flag==1)  
        cout<<"The number you entered is prime number";  
    else  
        cout<<"The number you entered is composite number";  
}
```

UNIT 7

POINTERS

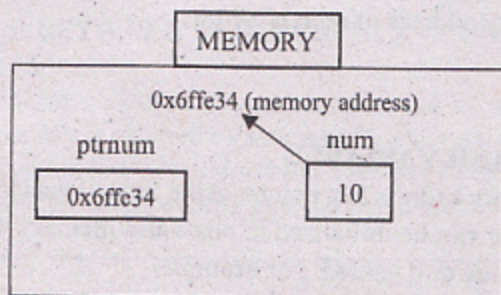
POINTER

Pointer variable is a variable that points to a specific address of other variable. It holds address of a variable. Pointers are used in a program to access memory and manipulate addresses.

MEMORY ADDRESS

Memory address is a unique number that is assigned to a memory location. The width of memory address depends on computer architecture. For example, a 16 bit computer architecture has 2^{16} memory locations. The pointer should be of 16 bits or 2 bytes of width to store this memory location.

When writing programs variables are needed to be declared. The declaration of variable tells the computer to reserve memory space for the variable. The name of the variable refers to that memory space (location), and that memory space or location has unique memory address. For example:



ADVANTAGES OF USING POINTER

- 1- The memory address can be accessed directly.
- 2- Memory can be saved.
- 3- As it does not duplicate data so it runs faster.

DECLARATION OF POINTER VARIABLE

The method of declaring a pointer variable is same as declaring a simple variable. The difference is that an asterisk * is used in the declaration. The data type of a pointer variable must be same as the data type of the variable whose memory address is to be stored in the pointer variable. The syntax of declaring pointer variable is as follows:

	datatype *variablename;	e.g	int *fptr;
OR	datatype* variablename;	e.g	int* fptr;
OR	datatype * variablename;	e.g	int * fptr;

REFERENCE OPERATOR (&) OR ADDRESS OPERATOR

Reference or Address operator is used to assign memory address to a pointer. It is denoted by & symbol. For example:

```
int *ptr;           // declaration of pointer variable 'ptr';
int num=40;         // constant 40 is assigned to variable 'num'
ptr=&num;            // memory address of variable 'num' is assigned to pointer 'ptr'
```

Example program to understand the concept of Reference operator.

```
#include<iostream>
using namespace std;
int main(void)
{
    int num=40;
    int *ptr;
    ptr=&num;
    cout<< "The value of num is"<<num<<endl;
    cout<< "The memory address of num is"<<ptr;
    return 0;
}
```

Output:

```
The value of num is 40
The memory address of num is 0x6ffc34
```

INITIALIZATION OF POINTER VARIABLE

The process of assigning a memory address to a pointer at the time of declaration is called pointer initialization. The pointer can be initialized to any valid memory address. It can also be initialized to a NULL value or 0 valued. For example:

```
float height;
float *fptr1=&height;
int *fptr2=NULL;
int *fptr3=0;
```

VOID POINTER

The void pointer is a type of pointer that can store the address of any type of variable. The keyword void is used as the data type of the void pointer as follows:

```
void *fptr;
float n1;
int n2;
fptr=&n1;
fptr=&n2;
```

DE-REFERENCE OPERATOR

De-reference operator is used to store or access the value of variable through pointer. It is denoted by asterisk (*).

Example program to understand the concept of De-Reference operator.

```
#include<iostream>
using namespace std;
int main(void)
{
    int num=40, valuenum;
    int *ptr;
    ptr=&num;
    valuenum=*ptr;
    cout<< "The value of num is "<<num;
    cout<< "\nThe memory address of num is "<<ptr;
    cout<< "\nThe value of ptr is "<<ptr;
    cout<< "\nThe value of *ptr is "<<*ptr;
    cout<< "\nThe value of valuenum is "<<valuenum;
    return 0;
}
```

Output:

```
The value of num is 40
The memory address of num is 0x6ffe30
The value of ptr is 0x6ffe30
The value of *ptr is 40
The value of valuenum is 40
```

DIFFERENCE BETWEEN DEREFERENCE OPERATOR '*' AND REFERENCE OPERATOR '&'.

Dereference operator is used to access the value of a variable through pointer. The dereference operator is denoted by asterisk (*). The reference operator also called address operator. It is used to provide address of a memory location when the address needs to be assigned to a pointer. It is denoted by ampersand (&).

EXERCISE**Q1. Select the best answer for the following MCQs.**

- If we have the statement `int *Ptr;` then to what Ptr point?
 - Point to an integer type variable.
 - Points to a character type variable.
 - Points to a floating point type variable.
 - None of above.
- A pointer is:
 - A keyword used to create variables.
 - A variable that stores address of instruction.
 - A variable that stores address of another variable.
 - All of above.
- The operator used to get value at address stored in a pointer variable is:
 - *
 - &
 - &&
 - ||
- Which of the statements is correct about the following segment code? `int i=10; int *j=&i;`
 - j and i are pointer to an int.
 - i is a pointer to an int and stores address of j.
 - j is a pointer to an int and stores address of i.
 - j is a pointer to a pointer to an int and stores address of i.
- In pointers, dereference operator(*)
 - Address the value of the pointer variable.
 - Points to the value stored in the variable pointed by the pointer variable.
 - Both a & b
 - None of the above.

6. If `double *p_total=&total;` what will be printed by the statement : `cout<<&p_total;`
a) Address of total b) Value of total c) Value of `p_total` d) Address of `p_total`
7. The statement `int *p;` has the same meaning as:
a) `int p;` b) `int* p;` c) `*int ptr;` d) `int p*;`
8. Which of the following assigns the address of `x` to the pointer `p1`?
a) `*p1=&x;` b) `p1=x;` c) `p1=&x` d) `&p1=*x`
9. Which of the following statement is invalid?
a) `int p=&t;` b) `int p= int *t;` c) `float t=&p2;` d) All
10. What does the following statement do? `double *total;`
a) Declares a pointer variable named `total` b) Initialize a variable named `*total`
c) Access the value of a variable `total` d) None

Q2. Write answers of the following questions.

1. What is pointer? Describe the advantages of using pointer variable.

Ans. See Ans on Page 119

2. What is the difference between the dereference operator `**` and reference operator `&`? Explain with the help of some lines of code.

Ans. See Ans on Page 120

3. How pointer is initialized? Write a simple program to illustrate this concept.

Ans. See Ans on Page 120

4. How the declaration of a pointer variable is different from the declaration of a simple variable.

Ans. The declaration of a pointer variable is simple and is similar to the declaration of a simple variable with a minor difference of the use of an asterisk(*) symbol between the data type and the variable name.

declaration of simple variable
`int a,b,c;`

declaration of pointer variable
`int *a,*b,*c;`

UNIT 7

CLASSES AND OBJECTS

CLASS

Classes are the most important features of Object Oriented Programming in C++. Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an object. The variables inside class definition are called as data members or attributes and the functions are called member functions. The functions are used to perform different operations on the attributes.

DECLARATION OF A CLASS

The keyword `class` is used to declare a class. The general format is as follows:

```
class class_name
{
    access_specifier_1:
    member1;
    access_specifier_2:
    member2;
    .
    .
    .
};
```

Body of class

Here `class_name` is a valid identifier for the class which is followed by the body of the class that is enclosed in pair of braces. The body of the class consists of:

- **Data members**

The variables defined inside a class are called data members. They are also called attributes. Mostly, the data members are declared under the **private** access specifier. It makes sure that the data of the class cannot be used directly outside from class.

- **Member functions.**

The functions defined inside a class are called member functions. The functions are used to perform different operations on the data members. Mostly, the member functions are declared under **public** access specifier. It makes sure that they can be directly used outside from class. For example:

```
class Test
{
    private:                // access specifier
        int a;
        char c;
        float x;
    public:
        display();
        input();
};
```

// declaration of data members

// declaration of member function

OBJECT

An instance or a variable of type class is known as an object. It holds data members and member functions defined in the class. The class does not occupy any memory for its members but the memory is allocated when the object of a class is created. The process of creating an object is called instantiating. An object is of great importance in Object Oriented Programming, because a class cannot be used without creating its object. The syntax for creating an object of a class is as follows:

```
class_name Object_name;
```

Suppose to create an object of class CRectangle we write:

```
CRectangle R1,R2,R3;
```

Here, R1, R2, and R3 are the objects of the same class CRectangle that share the same data members and member functions. To access members of an object, whether data members or member functions, dot operator (.) is used with the member name as follows:

```
object.member_name;
```

ACCESS SPECIFIERS

Access specifiers are used to specify access level of class members. In C++ access specifiers define how the members of the class can be accessed. They determine which member of a class is accessible in which part of the class/program. The most commonly used access specifiers in C++ are given below.

1) Private access specifier

2) Public access specifier

1) Private access specifier

Private access specifiers tells the compiler that the members defined in a class by preceding this specifier are accessible only within the class and its friend function. Private members are not accessible outside the class.

Example of using private access specifier

```
#include<iostream>
using namespace std;
class TestPrivate
{
```

```
    private:
```

```
    int x;
```

```
    public:
```

```
    void test()
```

```
    {    x=10;
```

```
        cout<<"x is private data member that is accessed inside the class only\n";
```

```
        cout<<" the value of x is "<<x<<endl;    }
```

```
};
```

```
int main(void)
```

```
{
```

```
    TestPrivate P1;
```

```
    P1.test();
```

```
    // P1.x=10; // it is an invalid statement because x is private that cannot be used from
                // outside the class.
```

```
    return 0;
```

```
}
```

Output:

```
x is private data member that is accessed inside the class only
the value of x is 10
```

2) Public access specifier

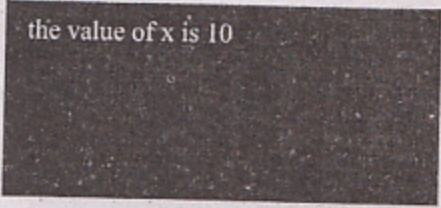
Private access specifiers tells the compiler that the members defined in a class by preceding this specifier are accessible from anywhere the object is visible i.e within the class, in the derived class and in the main function.

Example of using private access specifier

```
#include<iostream>
using namespace std;
class TestPublic
{
    public:
    int x;
    void show()
    {
        cout<< " the value of x is "<<x<<endl;
    }
};

int main(void)
{
    TestPublic P1;
    P1.x=10; // it is a valid statement because x is public that can be accessed from outside
            //the class.
    P1.show();
    return 0;
}
```

Output:



the value of x is 10

DATA HIDING

Data hiding is one of the important features of C++. It is the process of protecting class members against illegal access from outside the class. The data hiding is performed using different access specifiers such as private, public and protected. It is also known as Encapsulation.

The private access specifier is used to completely hide class members. The class members declared with **private access specifier** cannot be accessed outside the class. The class members declared with **protected access specifier** can only be accessed within the class or from the derived class. The class members declared with **public access specifier** can be accessed from anywhere in the program. Different levels of hiding the class members are as follows:

Access	Public	Protected	Private
Access of class members in the same class.	Yes	Yes	Yes
Access of class members in the same class.	Yes	Yes	No
Access of class members from outside the class.	Yes	No	No

CONSTRUCTOR

A constructor is a special type of member function that has same name as the class. It is automatically called when an object of that class is created. It is usually used to automatically initialize the data members. Constructors have specific rules/features which are as follows:

- 1) Constructors have the same name as that of the class.
- 2) Constructors have no return type (not even void).
- 3) Constructor is always public.

Example program to understand the concept of constructor.

```
#include<iostream>
using namespace std;
class number
{
    private:
    int x,y;
    public:
    number() // it is a constructor
    {
        x=50;
        y=60;
    }
    void avg()
    {
        cout<< "x="<<x<<endl;
        cout<< "y="<<y<<endl;
        cout<< "average="<<(x+y)/2<<endl;
    }
};
int main()
{
    number n; // the constructor number is called when the object n is created
    n.avg();
    return 0;
}
```

Output:

```
x=50
y=60
average=55
```

DESTRUCTOR

A destructor is a special type of member function that has same name as the class. It is automatically called when an object of that class is destroyed. The destructor works opposite to the constructor. It de-allocates the memory that is allocated to an object when it was created. The destructor name is preceded by tilde sign (~). Destructors have specific rules/features which are as follows:

- 1) Destructors have the same name as that of the class, must be preceded by tilde sign (~).
- 2) Destructors cannot accept parameters.

- 3) Destructors have no return type.
- 4) A class can have only one destructor.
- 5) A destructor cannot be overloaded.

Example of implementing a class with the name ConstDest have constructor and destructor functions in the body.

```
#include<iostream>
using namespace std;
class ConstDest
{
public:
    ConstDest()    // it is a constructor
    {
        cout<< "Constructor is called and executed\n";
    }
    ~ConstDest()  // it is a destructor
    {
        cout<< "Destructor is called and executed\n";
    }
};

int main()
{
    ConstDest a,b; // the constructor ConstDest as well as the destructor ConstDest is called
                  // when the object a and is created
    return 0;
}
```

Output:

```
Constructor is called and executed
Constructor is called and executed
Destructor is called and executed
Desstructor is called and executed
```

DEFAULT CONSTRUCTOR

The default constructor is a constructor that can be called with no arguments. It can be defined with no arguments or with default values given for all arguments. If we do not declare any constructor in a class, the compiler declares a default constructor with no arguments. The default constructor is also called implicit constructor.

USER DEFINED CONSTRUCTOR

The user defined constructor is a constructor that is defined by the user. The compiler does not declares any default constructor if the user-defined constructor already exists in the class. The user-defined constructor is also called explicit constructor.

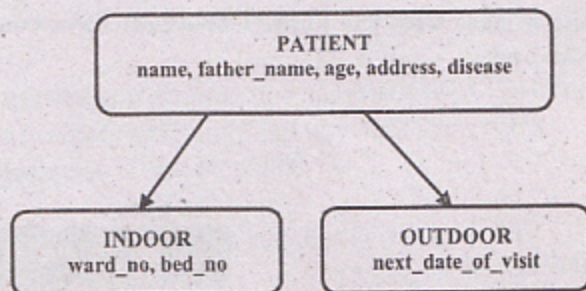
CONSTRUCTOR OVERLOADING

The process of using more than one constructors with same name but different parameters is known as constructor overloading. In constructor overloading, when the object is created, the compiler executes a constructor that matches the argument in function call.

INHERITANCE

A programming technique in which new classes are created from existing classes is called inheritance. It is a key feature of C++ language. Inheritance uses the concept of parent and child class. A **parent class** is the class from which other classes are derived. It is also

called **base class**. A **sub class** is a class which inherits all the properties and functions from base class. A **sub class** is also called child class or derived class. An example of inheritance from daily life is given below:



We can represent Patient class into indoor and outdoor patients in which both have some common features like name, father name, age, address and disease but indoor patient have 'ward_no' and 'bed_no' as its unique features and the outdoor patient have 'next_date_of_visit' as its unique feature. It can be represented as follows:

Syntax of using inheritance in classes

```

class sub_class: specifier parent_class
{
    body of the class;
};
  
```

For example:

```

class X           // base class
{
    body of class X;
};
class Y: public class X // class Y is derived from class X
{
    body of class Y;
};
  
```

Example program to understand the concept of inheritance.

```

#include<iostream>
using namespace std;
class X
{
    public:
    void showx()
    {
        cout<< "I am in base class X"<<endl;
    }
};
class Y:public X
{
  
```

Output:

```

I am in base class X
I am in derived class Y
I am in base class X
  
```

```

        public:
        void showy()
        {
            cout<< "I am in derived class Y"<<endl;
        }
    };

int main()
{
    Y y1;           //object of derived class Y
    X x1;
    y1.showx(); //function of class X can be used by object of class Y because it is inherited
    y1.showy();
    x1.showx();
    //x1.showy(); //it is invalid because function of Y cannot be used by object of X
    return 0;
}

```

POLYMORPHISM

Polymorphism is the ability to use an operator or function in multiple ways. Polymorphism gives different meanings or functionality to the operators or functions. Poly means many and morphism means forms. It means that there are many uses of these operators or functions. It uses a single function or operator in many ways. One of the daily life examples of polymorphism is that a person can be student as well as a friend to another person. Also a person can be an engineer as well as a teacher. In C++, polymorphism can be achieved by one of the following concepts:

- 1) Function Overloading 2) Operator Overloading 3) Virtual function

1) Function Overloading

Function overloading is the process of declaring multiple functions with same name but different parameters. The same function behaves in different ways when it receives different number or type of parameters.

2) Operator Overloading

Function overloading is the process of defining additional meanings of the operator. It enables an operator to perform different operations depending on operands. For example the addition operator (+) is used to add numeric values such as 20+10. The operator overloading can enable the additional operator (+) to concatenate (join) two strings such as "Pakistan"+"Zindabad".

2) Virtual Function

A function that is defined in the parent class and overridden in child classes is known as virtual function. Virtual functions are used to implement polymorphism. It is one of the important features of object-oriented programming.

EXERCISE**Q1. Select the best answer for the following MCQs.**

1. A constructor is called whenever _____
a) An object is destroyed. b) An object is created.
c) A class is declared. d) A class is used.
2. Destructor is used for _____
a) Initializing the values of data members in an object. b) Initializing arrays.
c) Freeing memory allocated to the object of the class. d) Creating Object.
3. The name of destructor is always preceded by the symbol :
a) + b) % c) - d) ~
4. Constructor are usually used for _____
a) Constructing programs. b) Running classes.
c) Initializing objects. d) All of the above.
5. Inheritance is used to _____
a) Increase the size of a program. b) Make the program simpler.
c) Provide the facility of code reusability. d) Provide the facility of data hiding.
6. A programming technique in which programs are written on the basis of objects is called:
a) Procedural programming b) Object-oriented programming
c) Non-procedural programming d) Query language
7. In C++, object of the class is also called?
a) Data collection b) Constants c) Instance d) Tag
8. The attributes of a class are called?
a) Data member b) Methods c) Member function d) Object
9. Data member is also called?
a) Attribute b) Method c) Class d) Object
10. In C++, the functions of a class are called?
a) Attribute b) Method c) Member function d) Object
11. The object of a class contains:
a) Data members b) Member functions c) Tag d) a and b
12. Which of the following are types of access specifiers?
a) sub and class b) virtual and function c) private and public d) null and void
13. _____ is a default access specifier for members of class in C++:
a) protected b) public c) private d) All
14. Members of objects are accessed with:
a) Resolution operator b) Address operator c) Dot operator d) Extraction operator
15. Which of the following specifiers is used to access class members outside the class?
a) public b) protected c) private d) default
16. The mechanism that binds code and data together and keeps them secure from outside the world is known as:
a) Abstraction b) Encapsulation c) Inheritance d) Polymorphism
17. Which of the following class inherits all the capabilities of the base class?
a) Abstract class b) Super class c) Parent class d) Child class
18. The derived class is also known as:
a) Parent class b) Child class c) Sub class d) Both b and c

19. The base class is also known as:

- a) Parent class b) Super class c) Child class d) a and b

20. If class A inherits from class B, then class B is called _____ of A.

- a) Parent class b) Sub class c) Abstract class d) Child class

Q2. Write answers of the following questions.

1. Explain the terms object and class with examples.

Ans. See Ans on Page 124

2. How objects are created to access members of a class?

Ans. See Ans on Page 124

3. What are access specifiers? Explain private and public specifier with examples.

Ans. See Ans on Page 125

4. What is data hiding? Explain.

Ans. See Ans on Page 125

5. Explain constructor and destructor class member functions with examples.

Ans. See Ans on Page 126

6. Explain inheritance and polymorphism with examples.

Ans. See Ans on Page 127-129

Lab Activities / Programs

1. Write a C++ program implementing a class with the name Circle having two functions with the names: GetRadius and CalArea. The functions should get the value for the radius from the user and then calculate the area of the circle and display the results.

Ans.

```
#include<iostream>
using namespace std;
class Circle
{
private:
    float rad;
public:
    void GetRadius()
    {
        cout<<"Enter Radius";
        cin>>rad;
    }
    void CalArea()
    {
        cout<<"\nArea of Circle is "<<3.14*(rad*rad);
    }
}
```

```
};  
int main()  
{  
    Circle c;  
    c.GetRadius();  
    c.CalArea();  
    return 0;  
}
```

2. Write a C++ program implementing inheritance between Employee (base class) and Manager (derived class).

Ans. #include<iostream>
using namespace std;
class Emp
{
private:
 int EmpID;
 int salary;
public:
 void GetEmp()
 {
 cout<<"Enter Employee ID ";
 cin>>EmpID;
 cout<<"Enter Employee Salary ";
 cin>>salary;
 }
 void ShowEmp()
 {
 cout<<"\nEmployees ID is "<<EmpID;
 cout<<"\nEmployees Salary is "<<salary;
 }
};
class Manager: public Emp
{
private:
 int DeptID;
public:
 void GetDept()
 {
 cout<<"\nEnter department ID ";
 cin>>DeptID;
 }
 void ShowDept()
 {
 cout<<"\nDepartment ID is "<<DeptID;

```
    }  
};  
int main()  
{  
    Manager m;  
    m.GetEmp();  
    m.GetDept();  
    m.ShowEmp();  
    m.ShowDept();  
    return 0;  
}
```

3. Write a C++ program implementing a class with the name ConstDest having a constructor and destructor function in its body.

Ans.

```
#include<iostream>  
using namespace std;  
class ConstDest  
{  
public:  
    ConstDest()  
    {  
        cout<< "Constructor is called and executed\n";  
    }  
    ~ConstDest()  
    {  
        cout<< "Destructor is called and executed\n";  
    }  
};  
int main()  
{  
    ConstDest a,b;  
    return 0;  
}
```

4. Write a C++ program implementing a class with the name Time. This class should have a constructor to initialize the time, hours, minutes and seconds, initially to zero. The class should have another function name ToSecond to convert and display the time pass by the object into seconds.

Ans.

```
#include<iostream>  
using namespace std;  
class Time  
{  
private:  
    int hr,min,sec;  
public:  
    Time()
```

```
{
    hr=min=sec=0;
}
void SetTime(int hh,int mm,int ss)
{
    hr=hh;
    min=mm;
    sec=ss;
}
void ToSecond()
{
    int s;
    s=sec+(min*60)+(hr*3600);
    cout<<"\nTime is "<<hr<<":"<<min<<":"<<sec;
    cout<<"\nTotal Seconds: "<<s;
}

};
int main()
{
    Time t1;
    t1.SetTime(1,5,8);
    t1.ToSecond();
    return 0;
}
```

UNIT 9

FILE HANDLING

FILE

A file is a collection of bytes stored on a secondary storage device like hard disk. The collection of bytes may be interpreted in different ways. For example, a text document may interpret the bytes as characters, words, lines, paragraph or pages. A database may be interpreted as fields and records. A graphical image may be interpreted as pixels. The meanings attached to a particular file is determined by the data structures and operations used by the program to process the file.

FILE HANDLING

In C++ file handling is a process of storing data permanently in computer. It provides a mechanism to write output of a program into a file or read from a file. The basic operations involved in file handling are as follows:

- Opening file
- Reading file
- Writing file
- Closing file

TYPES OF FILES

In C++ there are two types of files based on how they store data. These are:

- 1) Text files
- 2) Binary files

1) TEXT FILES

A type of file that stores data as readable and printable characters is called text file. A source program of C++ is an example of text file. The user can easily view and read the contents of a text file. It can also be printed to get a hardcopy.

Text files can be a stream of characters that can be processed by a computer sequentially in forward direction. That is why a text file is typically opened only for one type of operation at a time such as reading or writing etc. Only one character can be read or written to the text files at a time.

2) BINARY FILES

A type of file that stores data as non-readable binary codes is called binary file. An object file of C++ program is an example of binary file. It consists of binary codes that are converted by the compiler from source file. The user cannot read the contents of a binary file. It can only be read and processed by computer.

The binary files can be processed using sequential or random access methods. These files can be opened for read and write operations at the same time. For example, a database file is created and processed as binary file. The process of updating a record will be performed as follows:

- Locating the appropriate record.

- Reading the record into memory.
- Modifying the record as required.
- Writing the updated record back to the file.

FILE OPENING MODES

The file opening mode indicates the type of operations to be performed on the file after opening. A file can be opened in different modes using `open()` function. Different types of modes for opening a files are as follows:

Mode	Description
<code>ios::in</code>	Used to open a file for input operations (Reading a file).
<code>ios::out</code>	Used to open a file for output operations (Writing a file).
<code>ios::binary</code>	Used to open a file in binary mode.
<code>ios::ate</code>	Used to open a file for output operations and move the read/write control/position to the end of file.
<code>ios::app</code>	Used to open a file for append operation means the content will be appended/added at the end of file.
<code>ios::trunc</code>	Used to delete the previous contents of a file if the file is opened for output operations is already exists.

OPENING FILES

A file should be opened before it can be processed. A file can be opened by first creating an object of `ifstream`, `ofstream` or `fstream`. The object is then associated with a file. An open file is represented by a stream object in a program. Any input or output operation performed on this stream object is applied to the file associated with it. To open a file, the function `open()` is used whose syntax is as follows:

```
myfile.open(filename,mode);
```

Here "myfile" is an internal variable, actually an object, used to handle the file whose name is written in the parenthesis, "filename" is the name of the file that is to be opened, "mode" specifies the mode in which the file is to be opened.

The dot(.) operator is used between the variable `myfile` and `open` function. `myfile` is an object `open()` is a member function of `ifstream`. The argument for `open` function is the name of the file on the disk which should be enclosed in double quotes. The file name can be the simple file name like "student.txt". It can also be fully qualified path name like "C:\myprogs\student.txt".

To declare the variable used to handle the file, the following statement is used:

```
ifstream myfile;
```

DEFAULT OPENING MODES

The member function `open()` is available in all three classes `ofstream`, `ifstream` and `fstream`. The function uses default mode to open a file if mode parameter is not used by the user. The default modes of these classes are as follows:

Class	Default mode parameter
<code>ofstream</code>	<code>ios::out</code>
<code>ifstream</code>	<code>ios::in</code>
<code>fstream</code>	<code>ios::in ios::out</code>

For `ifstream` and `ofstream` classes, `ios::in` and `ios::out` are automatically and respectively assumed, even if a mode is not given as second argument to `open()` member function.

VERIFYING FILE OPEN

The stream object returns true value if the referred file is opened. It returns false if the file is not opened. An example is as follows:

```
ofstream myfile;  
myfile.open("verify.txt");  
if(myfile)  
cout<< "The file is opened;
```

OR

```
ofstream myfile;  
myfile.open("verify.txt");  
if(!myfile)  
cout<< "The file is NOT opened";
```

CLOSING A FILE

The opened file should be closed when the input or output operation on the file is finished. The member function `close()` of stream object is used to close a file. It doesn't have any parameters. The syntax to close a file is as follows:

```
myfile.close();
```

here "myfile" indicates the stream object that refers to a file. "close" is the name of member function that closes a file.

Example program that creates a text file "example.txt" and writes a line of text in it.

```
#include<iostream>  
#include<fstream.h>  
using namespace std;  
int main()  
{  
    ofstream myfile;
```

```
myfile.open("example.txt");
if(!myfile)
{
    cout<< "File is NOT opened";
    exit(1);
}
else
    myfile>> "Program to write into a file";
myfile.close();
return 0;
}
```

The code of above program creates a file called example.txt and inserts a sentence into it. A user can open this file and see the output.

Example program that reads one word from text file "example.txt" that is created in the previous program.

```
#include<iostream>
#include<fstream.h>
using namespace std;
int main()
{
    char ch[50];
    ifstream myfile;
    myfile.open("example.txt");
    if(!myfile)
    {
        cout<< "File is NOT opened";
        exit(1);
    }
    else
        myfile>>ch;
    cout<<"The first word of the file is "<<ch;
    myfile.close();
    return 0;
}
```

bof() and eof()

C++ provides two special functions bof() and eof() that are used to set the pointers to the beginning of a file or end of a file respectively.

The bof() function

The bof() function is used to find if the pointer is at the beginning of the file or not. It returns true if the control is at the beginning and returns false if it is not. The syntax of bof() function is as follows:

```
myfile.bof();
```

The eof() function

The eof() function is used to find if the pointer is at the end of the file or not. It returns true if the control is at the end. This function is very useful when you want to read records from a file that are unknown. The syntax of eof() function is as follows:

```
myfile.eof();
```

Example program that reads all the contents from text file "example.txt".

```
#include<iostream>
#include<fstream.h>
using namespace std;
int main()
{
    char ch[50];
    ifstream myfile;
    myfile.open("example.txt");
    if(!myfile)
    {
        cout<< "File is NOT opened";
        exit(1);
    }
    else

    while(!myfile.eof())
    {
        myfile>>ch;
        cout<< " "<<ch;
    }
    myfile.close();
    return 0;
}
```

STREAM

A stream is a series of bytes associated with a file. It consists of data that is transferred from one location to another. Each stream is associated with a specific file by using the open operation. The information can be exchanged between a file and the program once a file is opened.

TYPES OF STREAMS

Following are the two main types of streams in C++:

- 1) Input stream
- 2) Output stream

1) Input Stream

The input stream is used to input (read) data. It takes any sequence of bytes from an input device such as keyboard or file. Reading data from an input stream is performed using extraction operator >> with an object of istream or fstream. The ifstream is only used for input but fstream can be used for both input and output.

2) Output Stream

The output stream is used to output (write). It sends any sequence of bytes to an output device such as monitor or file. Writing data to an output stream is performed using insertion operator << with an object of ostream or fstream. The ofstream is only used for output but fstream can be used for both input and output.

TYPES OF STREAMS WITH RESPECT OF DATA READING AND WRITING

The data can be read from and written into files with the help of following streams:

- 1- Single Character Stream
- 2- String Stream

1- Single Character Stream

Using single character stream, the data can be read (input) from and written (output) to files character by character.

i- Writing File Character by Character

The put() function is used to write data character by character from files. The syntax to use put() function is:

obj.put(character_variable);
e.g. myfile.put(ch);

Example program that writes data character by character in the file named "charwrite.txt"

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
```

```
{
char ch;
int i;
ofstream myfile;
myfile.open("charwrite.txt");
cout<<"Enter character ";
    for(i=1;i<5;i++)
    {
        ch=getche();
        myfile.put(ch);
    }
myfile.close();
return 0;
}
```

ii- Reading File Character by Character

The `get()` function is used to read data character by character from files. When a character is read from a file, the pointer automatically moves to the next character, and when this function is used again then the next character is read. The syntax to use `get()` function is:

`obj.get(character_variable);`

e.g. `myfile.get(ch);`

Example program that reads data character by character in the file named "char.txt"

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
char ch;
ifstream myfile;
myfile.open("char.txt");
    while(!myfile.eof())
    {
        myfile.get(ch);
        cout<<ch;
    }
myfile.close();
return 0;
```

```
}
```

2- String Stream

Using String stream, the data can be read (input) from and written (output) to files as sequence of characters (string).

The `getline()` function

The C++ `getline()` is a standard library function that is used to read a string or a line from an input stream. It is a part of the `<string>` header.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char text[200];
    fstream file;
    file.open("example.txt", ios::out | ios::in);
    cout << "Write text to be written on file." << endl;
    cin.getline(text, 200);
    file << text << endl;    // Writing on file
    file >> text;            // Reading from file
    cout << text << endl;
    file.close();
    return 0;
}
```

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char str[20];
    fstream test;
    test.open("example.txt");
    while(!test.eof())
    {
        test.getline(str, 21);
        cout << str << endl;
    }
    test.close();
}
```

```
return 0;
}
```

EXERCISE

Q1. Select the best answer for the following MCQs.

1. eof() stands for _____.
a) errors-on-files. b) end-of-file. c) exit-of-file. d) none of above.
2. In file handling open() function is used _____.
a) to open C++ compiler. b) end-of-file. c) both a and b. d) none of the above.
3. Default mode parameter for ofstream is _____.
a) ios::out b) ios::in c) ios::binary d) both a and b
4. A text file has the extension _____.
a) .doc b) .docx c) .txt d) .bin
5. ios::binary is used as an argument to open() function for:
a) opening file for input operation. b) opening file for output operation.
c) opening file in binary mod. d) none of the above.
6. A file is stored in:
a) RAM b) Hard disk c) ROM d) Cache
7. Which of the following are file handling operations?
a) ios::in b) ios::out c) ios::app d) ios::truncate
8. The class ifstream, ofstream and fstream are defined in:
a) <iostream.h> b) <conio.h> c) <fstream.h> d) <ifstream.h>
9. Which of the following function is used to read data character by character from file?
a) write b) get c) put d) none
10. Which of the following function is the correct syntax to close a file in C++?
a) myfile\$close(); b) myfile@close(); c) myfile:close() d) myfile.close();

Q2. Write answers of the following questions.

1. What is file handling? Explain different types of file.

Ans. See Ans on Page 135

2. Describe different types of operations performed on the files.

Ans. See Ans on Page 135

3. Explain bof() and eof() functions.

Ans. See Ans on Page 139

4. Define streams. Describe input and output streams in detail.

Ans. See Ans on Page 139,140

5. What is meant by the term mode of file opening? Describe different modes of opening file.

Ans. See Ans on Page 136

ANSWER OF MCQs**Unit No. 1**

1) B	2) D	3) A	4) C	5) D	6) B	7) D	8) B	9) C	10) A
11) A	12) D	13) B	14) A	15) D	16) C	17) A	18) A	19) D	20) C

Unit No. 2

1) C	2) D	3) A	4) A	5) B	6) A	7) D	8) C	9) B	10) C
11) A	12) B	13) B	14) C	15) D	16) C	17) B	18) B	19) B	20) A

Unit No. 3

1) C	2) C	3) A	4) B	5) B	6) D	7) B	8) C	9) A	10) C
11) A	12) B	13) C	14) A	15) A	16) A	17) D	18) A	19) B	20) A

Unit No. 4

1) A	2) B	3) B	4) C	5) B	6) A	7) D	8) B	9) D	10) A
11) B	12) B	13) A	14) C	15) C	16) B	17) C	18) B	19) D	20) C

Unit No. 5

1) C	2) B	3) C	4) A	5) D	6) C	7) B	8) B	9) A	10) A
11) B	12) C	13) A	14) A	15) A					

Unit No. 6

1) C	2) C	3) A	4) A	5) D	6) D	7) D	8) B	9) D	10) D
11) C	12) B	13) D	14) D	15) D					

Unit No. 7

1) A	2) C	3) A	4) C	5) B	6) A	7) B	8) C	9) D	10) A

Unit No. 8

1) B	2) C	3) D	4) C	5) C	6) B	7) C	8) A	9) A	10) C
11) D	12) C	13) C	14) C	15) A	16) B	17) D	18) D	19) D	20) A

Unit No. 9

1) B	2) B	3) A	4) C	5) C	6) B	7) D	8) C	9) C	10) D

ABBREVIATIONS

1	OS	Operating System
2	SDLC	System Development Life Cycle
3	DOS	Dis Operating System
4	CLI	Command Line Interface
5	GUI	Graphical User Interface
6	I/O	Input Output
7	OOP	Object Oriented Programming
8	Cout	Console Output
9	Cin	Console Input
10	IOstream	Input output stream
11	Getch	Get character
12	Endl	End line
13	Setw	Set width
14	Strcpy	String copy
15	Strcat	String concatenation
16	Strlen	String length
17	Strrev	String reverse
18	Ifstream	Input file stream
19	Ofstream	Output file stream
20	Fstream	File stream
21	Bof	Beginning of file
22	Eof	End of file
23	DSL	Digital Subscriber Line
24	IBM	International Business Machines

COMPUTER SCIENCE (SYLLABUS)
NATIONAL CURRICULUM 2009
(For Grade 12)

UNITS	CONTENTS
UNIT 1 OPERATING SYSTEM	1.1 Introduction to Operating System i) Define an operating system ii) Describe commonly-used operating systems(DOS, Windows, Unix, Macintosh) iii) Explain the following types of operating system: <input type="checkbox"/> Batch processing Operating System <input type="checkbox"/> Multi-programming Operating System <input type="checkbox"/> Multi-tasking Operating System <input type="checkbox"/> Time -Sharing Operating System <input type="checkbox"/> Real-Time Operating System <input type="checkbox"/> Multi-processor Operating System <input type="checkbox"/> Parallel Processing Operating Systems <input type="checkbox"/> Distributed Operating Systems <input type="checkbox"/> Embedded Operating System iv) Define the following features/characteristics of operating system: <input type="checkbox"/> Single-user Operating Systems <input type="checkbox"/> Multi-user Operating System 1.2 Operating System Functions Describe the following main functions of operating system: <input type="checkbox"/> Process Management <input type="checkbox"/> Memory Management <input type="checkbox"/> File Management <input type="checkbox"/> I/O System Management <input type="checkbox"/> Secondary Storage Management <input type="checkbox"/> Network Management <input type="checkbox"/> Protection System <input type="checkbox"/> Command-Interpreter 1.3 Process Management i) Define a process ii) Describe the new, running, waiting/blocked, ready and terminated states of a process iii) Differentiate between: <input type="checkbox"/> Thread and process <input type="checkbox"/> Multi-threading & multi-tasking <input type="checkbox"/> Multi-tasking and multi-programming
UNIT 2 SYSTEM DEVELOPMENT LIFE CYCLE	2.1 System Development Life Cycle i) Define a System ii) Explain System Development Life Cycle (SDLC) and its importance iii) Describe objectives of SDLC iv) Describe stakeholders and their role v) Explain the following: <input type="checkbox"/> Planning <input type="checkbox"/> Feasibility

	<ul style="list-style-type: none"> <input type="checkbox"/> Analysis <input type="checkbox"/> Requirement Engineering <ul style="list-style-type: none"> Requirement Gathering Functional Requirements Non Functional Requirements Requirements Validation Requirements Management <input type="checkbox"/> Design (Algorithm, Flow Chart, Pseudo code) <input type="checkbox"/> Coding <input type="checkbox"/> Testing /verification <input type="checkbox"/> Deployment/Implementation <input type="checkbox"/> Maintenance/Support vi) Explain the role of following in the system development life cycle <ul style="list-style-type: none"> <input type="checkbox"/> Management <input type="checkbox"/> Project Manager <input type="checkbox"/> System Analyst <input type="checkbox"/> Programmer <input type="checkbox"/> Software Tester <input type="checkbox"/> Customer
<p style="text-align: center;">UNIT 3 OBJECT ORIENTED PROGRAMMING IN C++</p>	<p>3.1 Introduction</p> <ul style="list-style-type: none"> i) Define Program ii) Define header files and reserved words iii) Describe the structure of a C++ program <ul style="list-style-type: none"> <input type="checkbox"/> Pre-processor Directives <ul style="list-style-type: none"> include define <input type="checkbox"/> Main function <input type="checkbox"/> Body <p>iv) Know the use of a statement terminator (;)</p> <p>v) Explain the purpose of comments and their syntax</p> <p>3.2 C++ Constants and Variables</p> <ul style="list-style-type: none"> i) Explain the difference between constant and variable ii) Explain the rules for specifying variable names iii) Know the following data types offered by C++ and the number of bytes taken by each data type <ul style="list-style-type: none"> <input type="checkbox"/> Integer – int (unsigned, short, long) <input type="checkbox"/> Floating point – float <input type="checkbox"/> Double precision – double <input type="checkbox"/> Character – char <p>iv) Define constant qualifier – const</p> <p>v) Explain the process of declaring and initializing variables</p> <p>vi) Use type casting</p> <p>3.3 Input/ Output Handling</p> <ul style="list-style-type: none"> i) Explain the use of cout statement for displaying output on the screen ii) Explain the use of cin statement to get input from the keyboard during execution of the program iii) Define getch(), gets() and puts() functions iv) Define escape sequence v) Explain use of the following escape sequences using programming examples

	<input type="checkbox"/> Alert - \a <input type="checkbox"/> Backspace - \b <input type="checkbox"/> Newline - \n <input type="checkbox"/> Carriage Return - \r <input type="checkbox"/> Tab - \t <input type="checkbox"/> Display backslash - \\ <input type="checkbox"/> Display single quotation marks - \' <input type="checkbox"/> Display double quotation mark - \" vi) Make use of most commonly used I/O handling functions vii) Use manipulators endl and setw 3.4 Operators in C++ i) Define the following operators and show their use with examples: <input type="checkbox"/> Assignment operator (=) <input type="checkbox"/> Arithmetic operators (+, -, *, /, %) <input type="checkbox"/> Arithmetic assignment operators(+ =, -=, *=, /=, %=) <input type="checkbox"/> Increment and decrement operators (++ , --) - Prefix - Postfix <input type="checkbox"/> Relational operators (<, >, <=, >=, ==, !=) <input type="checkbox"/> Logical operators (AND, , OR &&, NOT !) <input type="checkbox"/> Ternary operator (? :) ii) Identify unary, binary and ternary operators iii) Define an expression iv) Define and explain the order of precedence of operators. v) Define and explain compound expression vi) Define compound expressions
UNIT 4 CONTROL STRUCTURES	4.1 Decisions i) Explain the use of the following decision statements: <input type="checkbox"/> If <input type="checkbox"/> If-else <input type="checkbox"/> Else-if <input type="checkbox"/> Switch-default ii) Know the concept of nested if iii) Use break statement and exit function 4.2 Loops i) Explain the use of the following looping structures: <input type="checkbox"/> For <input type="checkbox"/> While <input type="checkbox"/> Do-while ii) Use continue statement iii) Know the concept of nested loop
UNIT 5 ARRAYS AND STRINGS	5.1 Introduction i) Explain the concept of an array ii) Know how array elements are arranged in memory iii) Explain the following terms related to arrays <input type="checkbox"/> Size of array <input type="checkbox"/> Name of array <input type="checkbox"/> Index iv) Explain how to define and initialize an array of different sizes and data types v) Explain how to access and write at an index in an array

	vi) Explain how to traverse an array using all loop structures vii) Use the <code>sizeof ()</code> function to find the size of an array 5.2 Two dimensional Arrays i) Explain the concept of a two dimensional array ii) Explain how to define and initialize a two dimensional array of different sizes and data types iii) Explain how to access and write at an index in a two dimensional array 5.3 Strings i) Explain what are strings. ii) Explain how to define a string iii) Explain the techniques of initializing a string iv) Explain the most commonly used string functions
UNIT 6 FUNCTIONS	6.1 Functions i) Explain the concept and types of function ii) Explain the advantages of using functions iii) Explain the signature of function (Name, Arguments, Return type) iv) Explain the following terms related to functions <input type="checkbox"/> Function prototype <input type="checkbox"/> Function definition <input type="checkbox"/> Function call v) Explain the difference between local, global, and static variables vi) Explain the difference between formal and actual parameters vii) Know the concept of local and global functions viii) Use inline functions 6.2 Passing arguments and returning values i) Pass the arguments: <input type="checkbox"/> Constants <input type="checkbox"/> By value <input type="checkbox"/> By reference ii) Use default argument iii) Use <code>return</code> statement 6.3 Function overloading i) Define function overloading ii) Know advantages of function overloading iii) Understand the use of function overloading with: <input type="checkbox"/> Number of arguments <input type="checkbox"/> Data types of arguments <input type="checkbox"/> Return types
UNIT 7 POINTERS	7.1 Pointers i) Define pointers ii) Understand memory addresses iii) Know the use of reference operator (<code>&</code>) iv) Know the use of dereference operator (<code>*</code>) v) Declare variables of pointer types vi) Initialize the pointers

UNIT 8 CLASSES AND OBJECTS	8.1 Classes i) Define class and object ii) Know the member of a class: <input type="checkbox"/> Data <input type="checkbox"/> Functions iii) Understand and access specifier: <input type="checkbox"/> Private <input type="checkbox"/> Public iv) Know the concept of data hiding v) Define constructor and destructor <input type="checkbox"/> Default constructor/destructor <input type="checkbox"/> User defined constructor <input type="checkbox"/> Constructor overloading vi) Declare object to access <input type="checkbox"/> Data members <input type="checkbox"/> Member functions vii) Understand the concept of following only with daily life examples: <input type="checkbox"/> Inheritance <input type="checkbox"/> Polymorphism
UNIT 9 FILE HANDLING	9.1 File Handling i) Know the binary and text file ii) Open the file <input type="checkbox"/> Modes of opening file iii) Know the concept of <input type="checkbox"/> BOF <input type="checkbox"/> EOF iv) Define stream v) Use the following streams <input type="checkbox"/> Single character <input type="checkbox"/> String

PRACTICALS**Activities for Grade 12**

1. Installation of C++ Compiler
2. Familiarization with IDE of C++ Compiler
3. Write some programs using:
 - ☐ Cin
 - ☐ Cout
 - ☐ Escape sequences
 - ☐ Setw
4. Write program for problems like:
 - ☐ Solving arithmetic problems to (calculate interest, percentage, average, ratio, grades etc.)
 - ☐ Calculating area / volume / perimeter of some basic geometrical shapes
 - ☐ Comparing numbers / strings
 - ☐ Solving quadratic equation
 - ☐ Finding out the GCD and LCM.
 - ☐ Reading a number and find out whether it is a prime or composite
 - ☐ Sorting a list of items (numeric / string)
 - ☐ Searching an item out of a list of items (numeric /string)
 - ☐ Generating random numbers for a dice using function
 - ☐ Finding addition and multiplication of a matrices (Maximum 3 x 3)
 - ☐ Finding the transpose of a matrix (3 x 3)
 - ☐ Generating and summing simple series
 - ☐ Reversing a given number / string
 - ☐ Finding out a specific day of a week for a given data using function.
5. Write a programme to sum two and three numbers of different data types
6. Write a programme to display the address and the value of a variable using pointer
7. Write a programme to create and display student object with data members as name, age and class.
8. Write a programme to create and read a data file

Unit-wise Weightages – FOR PAPER

Unit	Title	Weight-age %
1	Operating System	10%
2	System Development Life Cycle	10%
3	Object Oriented Programming Using C++	10%
4	Control Structure	15%
5	Arrays and Strings	15%
6	Functions	15%
7	Objects and Classes	10%
8	Pointers	5%
9	File Handling	10%

**"FUNDAMENTALS OF COMPUTER &
DATABASE CONCEPTS" for GRADE 11 by
Sheikh Faisal Manzoor is also available at
bookshops.**

For Suggestions
email: misterfaisal.fg@gmail.com

**For Computer lectures for Grade 12, DON'T FORGET TO SUBSCRIBE
THE YOU CHANNEL:**



Sheikh Faisal Manzoor

Computer Science Lectures for 2nd year

www.youtube.com

<https://www.youtube.com/channel/UCAuBv05i0IGaHNFwt9s1z8A>

“Computers are incredibly fast, accurate and stupid;
humans are incredibly slow, inaccurate and brilliant;
together they are powerful beyond imagination.”

Albert Einstein.

Available at:

<i>1. Abdul Ghafoor Stationary Mart, Archer Road Quetta, Ph 0812842180</i>
<i>2. Al Khair Book Seller, Archer Road Quetta, Ph 0812848176.</i>
<i>3. Ashraf Stationary Mart, Urdu Bazar Archar Road Quetta, Ph 0812867676.</i>
<i>4. Bashir Book Depot, Mecongy Road near Dr Yaseen Clinic, Ph081 2661854.</i>
<i>5. Book City, Shop No 3 Gulistan Road Quetta, Ph 0812836644.</i>
<i>6. Books n Books, Shop No: 56 Chiltan Market Quetta Cantt.</i>
<i>7. Bookish , Gulistan Road Quetta Cantt, Ph 0812832223.</i>
<i>8. Imran Stationers, Archar Road near Islamia Masjid Quetta.</i>
<i>9. Mr. Book, Gulistan Road Quetta.</i>
<i>10. New Iqra Books & Stationers, Shop No 9 Chiltan Market Quetta Cantt.</i>
<i>11. Public Stationers, Urdu Bazar Archar Road Quetta, Ph 2841611.</i>
<i>12. Saad Book Bank Archar Road Quetta, Ph 2826723.</i>

STOCKIST

HASSAN BOOK PALACE
ARCHAR ROAD QUETTA, PH 0812867691